Recommending Heterogeneous Resources for Science Gateway Applications based on Custom Templates Composition

Ronny Bazan Antequera, Prasad Calyam, Arjun Ankathatti Chandrashekara, Reshmi Mitra
University of Missouri-Columbia, USA
Email: *{rcb553, aacwb} @mail.missouri.edu, {calyamp, mitrare}@missouri.edu*

Abstract

Emerging interdisciplinary data-intensive science gateway applications in engineering fields (e.g., bioinformatics, cybermanufacturing) demand the use of high-performance computing resources. However, to mitigate operational costs and management efforts for these science gateway applications, there is a need to effectively deploy them on federated heterogeneous resources managed by external Cloud Service Providers (CSPs). In this paper, we present a novel methodology to deliver fast, automatic and flexible resource provisioning services for such application-owners with limited expertise in composing and deploying suitable cloud architectures. Our methodology features a Component Abstraction Model to implement intelligent resource 'abstractions' coupled with 'reusable' hardware and software configuration in the form of "custom templates" to simplify heterogeneous resource management efforts. It also features a novel middleware that provides services via a set of recommendation schemes for a context-aware requirement-collection questionnaire. Recommendations match the requirements to available resources and thus assist novice and expert users to make relevant configuration selections with CSP collaboration. To evaluate our middleware, we study the impact of user preferences in requirement collection, jobs execution and resource adaptation for a real-world manufacturing application on Amazon Web Services and the GENI cloud platforms. Our experiment results show that our scheme improves the resource recommendation accuracy in the manufacturing science gateway application by up to 21% compared to the existing schemes. We also show the impact of custom templates knowledgebase maturity at the CSP side for handling novice and expert user preferences in terms of the resource recommendation accuracy.

Keywords: `federated cloud resources; component abstraction model; custom templates; novice and expert user preferences; cloud resource recommendation scheme`

1. Introduction

Data-intensive science gateway applications in fields such as bioinformatics, climate modeling, and particle physics are becoming increasingly popular. These science gateway applications present unique requirements in terms of deployment of distributed heterogeneous infrastructure and use of advanced cyberinfrastructure technologies/protocols such as high-speed data transfer protocols, end-to-end virtualization, local/remote computing, Software-defined Networking (SDN) with OpenFlow support, Network Function Virtualization (NFV), end-to-end performance monitoring [1], and federated identity & access management [2]. In most cases, such resources are shared among federated user sites and are handled as 'component' solutions that can be combined for transforming a local applications (frequently at desktop scale) to a hybrid cloud application (i.e., infrastructure composed of distributed/federated cloud resources) [3].

Traditional infrastructure deployment approaches use a five-step waterfall model [4] involving: sequential abstraction, analysis, component deployment, recursive benchmarking and infrastructure deployment steps. Such a tedious five-step approach generally requires involvement of experts to accurately identify the application requirements, compose feasible solutions, and re-tune the components post-deployment to improve upon sub-optimal outcomes. Alternatively, automation of such a waterfall model through expert systems can be performed using suitable 'abstractions' of heterogeneous resource components and a 'reusable' solution model. The lack of abstraction and reusability of configurations in the approaches makes provisioning of heterogeneous resources for data-intensive applications quite time-consuming and prone to guesswork. This subsequently leads to sub-optimal and cost-prohibitive outcomes for data-intensive application users and can impede wide-adoption of dynamic distributed resource management.

There are existing infrastructure-level automated resource deployment approaches that offer automation and reusability to some extent. These solutions include: VMware Appliance [5] (vApps) from VMware; National Science Foundation (NSF) sponsored cloud Global Environment for Network Innovations (GENI) RSpecs [6] using XML files; and Amazon Machine Images [7] by Amazon Web Services (AWS). However, due to their proprietary nature, they assume a homogeneous deployment environment that features only their API (Application Programming Interface) that hides the complexity, and thereby, obviates the need for abstractions to facilitate heterogeneous resource provisioning. There are also existing application level

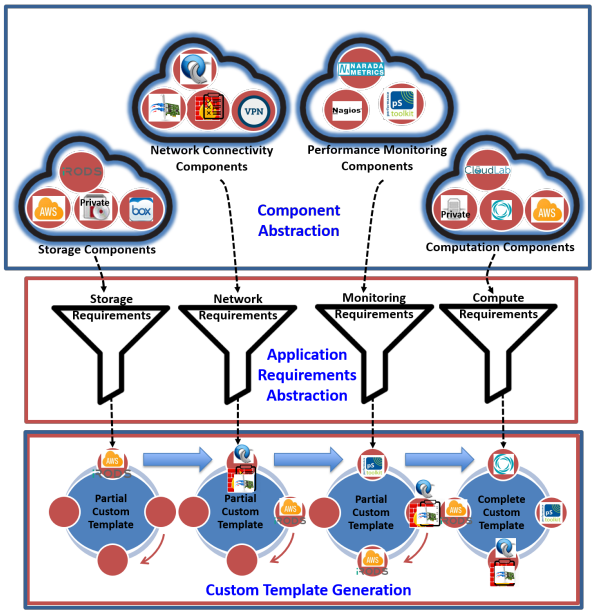


Figure 1: Abstraction of federated cloud infrastructure components to compose solutions that integrate heterogeneous cloud resources.

approaches such as, VMware ThinApp, Citrix XenApp and Microsoft App-V that are limited in terms of platform and operating system independence. In addition, these technologies are mostly targeted towards commercial use. Due to issues such has portability and licensing, they are yet to gain wider acceptance within the data application owners and science gateway community. Thus, a suitable abstraction, virtualization, and orchestration approach is needed to fully comprehend the reusable heterogeneous distributed resource deployments for data-intensive applications for science gateways.

In this paper, we build upon our recent prior work in [8] in order to address the above limitations of abstraction efforts and deployment approaches for heterogeneous distributed computing infrastructures. More specifically, we present a novel middleware approach shown in Figure 1, for integrating federated/distributed cloud resources to support data-intensive application user needs. The goal of the design of our middleware is to enable it to create data-intensive science gateway application resource requirement 'abstractions' to foster pertinent cloud resources recommendations, coupled with 'reusable' approaches and automated resource deployment to save time and effort. Our middleware implements the *Component Abstraction Model* which is designed to abstract and group different cloud resources into categories of heterogeneous resource components through a corresponding *Application Requirement Abstraction*. It generates reusable, and extensible *Custom Templates* of components to fulfill the diverse data-intensive application requirements. Our middleware builds upon existing technologies, and protocols and uses a "Custom Template Catalog" for storage and reuse of these Custom Templates.

Our middleware features a *Recommender system* with a set of recommender schemes that enable the reuse of existing custom templates. An offline initial recommender module improves the user productivity by narrowing down cloud resource composition to the most appropriate options. In addition, an online iterative recommender module regularly monitors the application behavior, and dynamically provides automated resource adaptions based on application demands. The adaptations can be fine-grained changes to an existing configuration that involves e.g., adjusting the virtual machine (VM) count. Alternately, adaptations can be coarse-grained changes that involve recommendation of a completely new custom template. Lastly, the chosen template can be provided to a *Resource Deployment Engine* that uses CSP-specific APIs, Docker technology [31] (other frameworks such as Terraform engine [9] could be substituted) and automates the process for cloud resources deployment in a federated-distributed cloud environment.

We evaluate our middleware recommendation scheme with simulated interactions and a real-world data-intensive case study in the Advanced Manufacturing domain scope. We simulate a series of user interactions for diverse applications requirements, considering 'novice users' (users who provide reduced or incomplete information to the KIS) and 'expert users' (users who are more aware of their data-intensive application requirements and provide more inputs to the KIS). Next, our evaluation of the middleware implementation features a real-world data-intensive manufacturing science gateway application with computing workflow requirements involving a cluster of systems/nodes. Our experiment results demonstrate significant improvement for novice/expert users to effectively express their data-intensive application requirements to the KIS, and subsequently access federated cloud resources that are automatically deployed. This whole process reduces the resource provisioning time drastically and the guesswork in selecting appropriate cloud resource allocations. We also show the benefits of using dynamic scaling knowledge to motivate application adaptations in terms of performance, agility and cost factors.

For our experiment scenarios, we use the Amazon Web Services (AWS) and GENI cloud [6] platforms as part of a public-private cloud testbed with distributed heterogeneous resources that can be discovered and configured by using the `geni-lib`, a GENI API capability. Through extensive simulations, we quantify the accuracy metric in terms of the percentage of satisfied user requests for a set of novice/expert user requirements. Accuracy is measured with respect to the increase in the amount of missing infrastructure requirement parameters within user preferences, and is also based on the catalog knowledgebase maturity. The simulation results demonstrate that the increase in catalog size results in an increase of the search space complexity, and consequently further hinders the accuracy of the resource recommendation.

The reminder of the paper is organized as follows: Section 2 presents related works. Section 3, describes our Component Abstraction Model. In Section 4, we present our Recommender Algorithm. In Section 5, we present details of our middleware implementation and resource deployment methodology. Section 6 discusses the performance evaluation and Section 7 concludes the paper.

2. Related Work

2.1. Expression and abstraction of user requirements

It is important to identify and abstract application requirements, as well as an available pool of resources to effectively provision infrastructure configurations. Hence, our literature review commences with the study of application requirement abstraction for infrastructure configurations to effectively provision resources. The authors in [10] divide the requirements into independent components and also use component abstraction to represent the knowledge. They break down a given problem by using heuristic rules in order to plan at the component level. Planning is done within each components to select potential and more suitable options and across the components to finally have a solution which combines options from every component. In [11], the authors use abstractions for component-based software architectures by estimating the assembly of reusable software components. The abstractions also involve making a forecast on software properties to an associated architecture by using graph-theoretic approaches. In [12], authors apply abstractions to check large asynchronous designs based on component abstraction verification in isolation. Similarly, authors in [13] present different software engineering theorems for a hierarchical abstraction model pertaining to knowledge development in the brain.

Existing abstraction models and methodologies can be leveraged as initial approaches for application requirement and cloud infrastructure abstraction of distributed heterogeneous resource, an area which we found is still quite under-explored. Our prior work [8] adapted the idea of component level abstraction to abstract distributed heterogeneous resources by organizing the available cloud resources into different components such as networking component, storage component, computation component etc. as shown in Figure 1. Thus, the decisions are made within and across the components, which makes resource planning and composition relatively easy and more efficient. In this paper, we propose a dynamic and context aware questionnaire which is an improvement over a static set of questions for all the application users, irrespective of their understanding about the CSP infrastructure capabilities. Our questionnaire approach is driven by pre-defined rules provided by a domain expert, and can help in the requirements collection for both novice and expert users.

2.2. Cloud service matching and recommendation

In recent times, a considerable amount of research work has been done on the cloud service matching problem and the related resource recommendation problem. Given the user requirements, the service matching algorithms should find the most suitable cloud resources from multiple CSPs. The most suitable resource is based on the user's importance of the functional requirements (e.g., CPU, memory, storage capacity, network bandwidth) or the non-functional requirements (e.g., service response time, frame rate, cost) of the resources. Although there are many available CSPs, application-owners with limited expertise find it challenging to compose and deploy their data-intensive science gateway applications on suitable cloud architectures. Works such as [14] – [19] provide guidance about helping a user specify the functional requirements such as e.g., VM-type for the desired QoS.

Filtering or comparing the resources based on a given set of functional and/or non-functional requirements is a challenging problem, and prior research efforts formulate this as a multi-criteria decision making (MCDM) problem. For example, authors in [20] propose a recommendation system in the form of a vector with its elements consisting of cost, performance, and mobility generated for available resources. This is matched to the vector denoting user requirements, with the goal of finding resource similarity. In [21], authors have developed a mathematical model to map user requirements onto the cloud resources considering e.g., cost, pricing policy, performance; they rank them according to the best resources for a given user's specification. Authors in [22] developed a comprehensive user interface for cloud service selection based on multiple criteria and considered user preferences (i.e., cost, QoS attributes). However, their approach lacks a deployment engine and functions as the equivalent of an e-commerce website for a CSP. In comparison, our work considers a set of recommender schemes that account for several variables such as cost, QoS and feature importance that should be taken into account as they need to be customized depending on user preferences as well as application resource demands.

All of the above works address the problem of finding, selecting or comparing the available resources of a single CSP e.g., they consider only single VM or just the storage or compute resource dimensions of a CSP, and rank them according to users QoS and/or functional requirements. In real world scenarios, user requirements are not just limited to adaptations that involve changes across a single VM or a related storage service. Instead, they need a collection of heterogeneous resources that can dynamically accommodate application demands and yet, operate within the cost and agility offerings of the CSP. There are few works which address the problem of collecting the user requirements in order to create a *comprehensive custom template* knowledgebase, and deploy the selected resources on relevant cloud platform architectures.

AWS Cloud Formation [23] from Amazon describes the cloud resources using templates. Similarly, Cisco UCS Director [24], a Unified Infrastructure Management Middleware, also provisions compute, network and storage resources automatically in local and remote locations using a similar template scheme viz., 'workflows'. Although both Cisco and Amazon abstract the user requests with the concept of templates, however they are proprietary in nature and are not portable. In contrast, we build templates according to Topology and Orchestration Specification for Cloud Applications (TOSCA) [25] standards. TOSCA is an OASIS standard language to describe a topology of cloud based web services, their components, relationships, and the processes that manage them. It includes specifications to describe processes that create or modify web services independent of the cloud service provider. A cloud provider for instance, could use TOSCA to define and com-

pose a specific cloud service. TOSCA compliance from a cloud provider perspective helps users to standardize how application deployments are configured with dependency/resource information in a cloud agnostic as well as portable manner. This enables the TOSCA templates to describe cloud resource configurations that involve multi-cloud resource specifications. The notable issue in the existing tools is that they lack a recommendation system and require the user to be aware of CSP offerings. Our solution approach features custom templates, which are not just limited to a single VM with software installation details, but contain specifications for a cloud architecture and related resources configuration in accordance with the TOSCA standards.

QuARAMRecommender [26] that is closely related to our work uses case-based reasoning for resource recommendation to suggest the CSP and VM type. It uses available cloud resources, service information and a knowledgebase to store previously recommended solutions. The knowledgebase has a requirements part and a solution part. For any new user request, the knowledgebase is searched using the k-Nearest Neigbhors (KNN) algorithm to get the solutions with similar requirements. In addition, there propose an adaptation module which modifies the fetched solution to better suit the new (user) request. Their implementation also has monitoring module which checks for any SLA violations. If there is one, a corresponding template in the Knowledgebase is given low priority and this decreases the chances of its retrieval in the future. Although, they use templates and a knowledgebase which stores the previous solution(s), the stored information is only limited to a VM type and a CSP recommendation. The template concept is also focused only on the configuration of the software packages that need to be installed on a VM. Furthermore, we extend the work in [26] by storing the templates in a Catalog (i.e., a Knowledgebase) for its reuse and also a recommender system which finds the suitable architecture for science gateway application on available cloud platforms. The user requirement collection step is very important and we have developed it as a rule based interaction questionnaire to obtain the user preferences. Our requirement collection is not restricted to a static set of questions, instead it flows dynamically in an interactive manner where answers to current set of questions trigger the selection of the next set of questions.

2.3. Automated resource provisioning frameworks

Deployment services should be independent of the CSP to support portability. There are few technologies and studies that focus on deployment of application on cloud platforms. Authors in [27] present a web framework which extracts the arbitrary program of an application from GitHub and deploys it in a user-desired cloud. The toolset converts the application to be deployed on the cloud into virtual machine images. However, this solution is restricted in its ability to execute tasks in standalone systems and is not generalized to all kinds of applications. In [28], authors describe an architecture composed of four layers (Cloud infrastructure, Abstraction, Orchestration, and Design) that enables automated deployment of cloud services. It has templates that encapsulate resource requirements and a deployment layer which provisions the resources described in the template on multi-cloud platform. However, similar to AWS Cloud formation and Cisco UCS Director, this is a descriptive way of collecting user requirements where the user has to be aware of the cloud resources and services available. In contrast, our work frees the user from such a situation, and our approach extracts the application requirements using context-awareness and uses this information to recommend suitable templates on available cloud platforms.

In [29], the authors present MetaConfig system, a tool similar to Puppet [30] that integrates configuration management, virtual machine allocation, and bootstrapping for virtual machine allocation. The MetaConfig system is flexible enough to include unexpected changes and scalability requirements, however, it does not consider configuration customization before the system is deployed. Our work also considers container technology such as Docker [31] to conduct reproducible experiments for data-intensive applications. Many works such as [32, 33] have studied the benefits of using Docker for HPC computing and scientific workflow deployments, and have found it to be a feasible solution for workflow reproducibility in cloud platforms for science gateway applications [34]. By "dockerizing" the whole application, it is relatively easy to move to different environments in cases of CSP infrastructure changes.

Some studies on middleware to provision elastic resources for cloud application have been developed such as Celar [35] that allows users to select and deploy certain compute resources in different cloud platforms through API calls using abstraction libraries (such as Apache Libcloud [36], jclouds [37], deltacloud [38]). Tools such as Celar requires code to describe the infrastructure using libraries. Such an approach is not user friendly and moreover those generalized libraries are not mature enough to handle all the resource provisioning requirements. Our middleware can be integrated into emerging heterogeneous distributed computing infrastructures and can be extended to complement other frameworks such as Cloudify [39] that choose to adopt related standards such as OASIS TOSCA [25].

3. Component Abstraction Model

In this section, we present the details of our component abstraction model solution. Our aim is to transform the resource performance expectations of data-intensive applications' (through intelligent requirements collection) into reusable resource recommendation templates. For this, we develop a Custom Template Cycle shown in Figure. 1 that involves three stages: Collection, Composition and Consumption. Figure 2 presents the Custom Template Life Cycle stages and the intermediate steps involved. The life cycle stages are, as follows:

- **Collection:** In this stage, we collect and abstract the data-intensive application requirements and categorize them into different resources domains.
- **Composition:** Depending on the requirement of the data-intensive application, we recommend either existing similar custom template solutions from pre-existing templates or compose new custom templates.

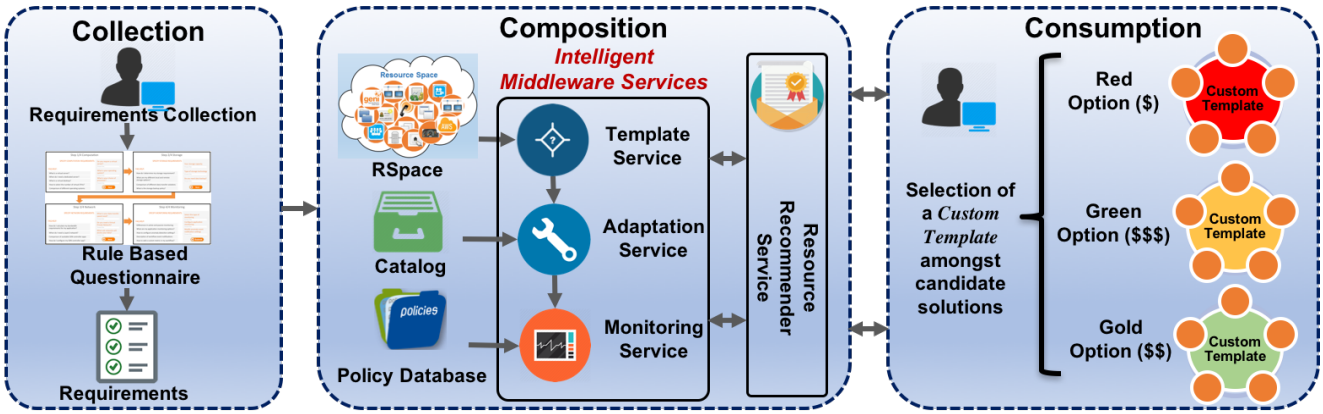


Figure 2: Custom Template Life Cycle. Our middleware solution presents three main stages: (a) Collection, that abstracts data-intensive application requirements. (b) Composition, that recommends template solutions based on the comparison of requirements with templates stored in a catalog or by the creation of new templates, and (c) Consumption, that enables cloud resource deployment automation.

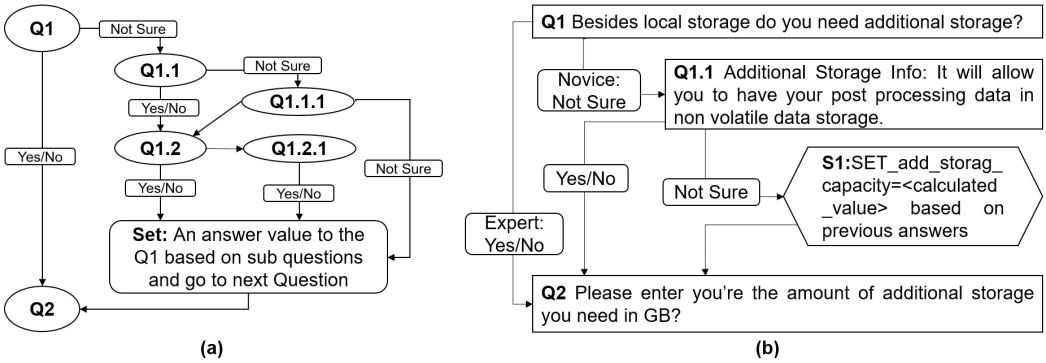


Figure 3: Rule based questionnaire generation, where answers to the previous or initial set of questions determine the next set of question

- **Consumption:** Candidate custom templates configurations are presented to the data-intensive application user who is a novice or an expert. Upon selection, the resource allocation process automatically deploys cloud resources along with required software for user access.

3.1. Collection

This is the first step where we collect and abstract user requirements through the Knowledge Interface System (KIS), which is a web based questionnaire portal. Traditionally, the requirement collection is done through one-to-one interviews and meetings between users and cloud engineers. This website mimics the interaction scenarios for infrastructure requirement elicitation process with domain expert that correspond to first set of steps in Figure 2. Our questions can be organized into sections for example: General information, Networking, Storage, Computation and Software requirements. Each section gathers specific domain information that helps to understand application infrastructure and software needs and recommends adequate solutions that might range from virtual machines with standard OS capabilities to distributed resources over federated CSP software that needs heterogeneous cloud resources.

The main function of the KIS is to collect user requirements for their applications and to help novice users to choose compatible resources. The initial set of questions aim to collect applications' high-level requirements and preferences to understand resource priority needs. This collected information not only helps to generate next set of questions but also helps the KIS to pre-populate relevant fields with default values. However, users (novice and expert) can always change the pre-populated values to personalize their solution.

The pre-population of fields assumes that the KIS design has a deep understanding of the underlying application community knowledge (e.g., in advanced manufacturing and bioinformatics). For instance, the KIS design requires close collaboration with the application community, and a community domain expert needs to input pertinent rules that are essential to select the most adequate answer. This input can also be inputs from an expert user during previous interactions with the KIS, and thus we can provide minimum functional resource compatibility for a given set of application requirements within a user community. The KIS additionally allows users to reuse previously deployed templates (select or upload template) in order to customize them and skip the questionnaire process.

It is possible for the KIS robustness in the questionnaire generation to be affected by wrong or incomplete information especially given by novice users. This is indeed a challenging problem to detect and cope; the extent of robustness in our KIS is based on the rules defined by an application expert in

our policy database (explained later in Section 3.2.3) used for the user requirements collection. For every question, the user has the option to choose a "Not Sure" response, choose default (pre-populated) values obtained from the catalog, or even just leave the answer field blank. When user intentionally gives wrong requirements, the deployment will result in either over- or under-provisioning of the resources, which in turn could lead to increased application cost or decreased application performance, respectively. To guide the user in such drastic cases, our KIS approach relies on two strategies: First, novice users, who are more likely to give incomplete/wrong specifications, are provided with more questions and guided documentation (e.g., terms definition, default parameters) based on their responses in terms of deployment options to compensate for their lack of knowledge. Secondly, major discrepancies in user requirements will be detected by the resource monitoring engine (as detailed later in Section 4.3) from the utilization data and its correlation with the original user requirements provided to the KIS. In such cases, the short-term adaptation of the resource provisioning is performed during an online recommendation stage facilitated via the KIS.

Additional structures that map the application requirements and the corresponding resource deployment are: Application Requirement Identifier (ARI), Resource Space (RSpace) and Macro Operators (MacOps), whose structure details are summarized below based on details from our prior work [8].

3.1.1. Application Requirement Identifier

When users finalize their interactions with the KIS, the collected data is automatically translated into an Application Requirement Identifier (ARI) structure. An ARI structure represents the meta-information of an application's resource requirements containing a vector consisting of cloud resources *Features* and corresponding *Preconditions* that satisfy the deployment of the application. A typical ARI structure has F_i that represents required resource feature (i.e., Bandwidth, Storage Size, RAM size) and P_i represents the corresponding required precondition (i.e., 20 Mbps bandwidth, 20 GB HDD storage, 8 GB RAM). A single or multiple such $F_i - P_i$ pair/s ($i \in \mathbb{I}$, where $\mathbb{I}$ is the set of required resources, and $I = |\mathbb{I}|$ is the number of elements in $\mathbb{I}$) that may belong to any new user application A_k. Satisfying an application's resource requirements is subjected to the successful fulfillment of all the features' preconditions.

3.1.2. Resource Space

The collected data is then compared with the cloud resource components called Resource Space (RSpace). The components within the RSpace are categorized into domains, i.e., VPN (virtual private network) and network bandwidth resources belong to the overall Network Connectivity domain and number of CPU cores, RAM size belong to the overall Computation domain. We define $\mathbb{D}$ as the set of all cloud resource domains where $\mathbb{D}$=$\{D_1, D_2, ..., D_N\}$, and N is the number of domain categories the cloud resources are divided into N=$|\mathbb{D}|$. Each such domain consists of a different number of cloud resources as follows:

$$
\begin{aligned}
D_1 &= \{R_1^1, \cdots, R_{\mathbb{A}}^1\} \text{where } \mathbb{A} \text{ is the no. of resources in } D_1 \\
D_2 &= \{R_1^2, \cdots, R_{\mathbb{B}}^2\} \text{where } \mathbb{B} \text{ is the no. of resources in } D_2 \\
&\vdots \\
D_N &= \{R_1^N, \cdots, R_{\mathbb{N}}^N\} \text{where } \mathbb{N} \text{ is the no. of resources in } D_N
\end{aligned}
$$

Each such resource consists of one or many resource *specifications* and their corresponding *constraints*, which define the performance bounds of the respective resources and the associated cost. For simplicity, we are not showing cost associated with the resource in the representation shown below. The RSpace data structure is represented as:

$$
\begin{aligned}
R_1^1 &= \{(S_{11}^1, C_{11}^1), \cdots, (S_{1\mathcal{A}}^1, C_{1\mathcal{A}}^1)\} \\
R_2^1 &= \{(S_{11}^2, C_{11}^2), \cdots, (S_{1\mathcal{B}}^2, C_{1\mathcal{B}}^2)\} \\
&\vdots \\
R_{\mathbb{A}}^1 &= \{(S_{11}^{\mathbb{A}}, C_{11}^{\mathbb{A}}), \cdots, (S_{1\mathcal{N}}^{\mathbb{A}}, C_{1\mathcal{N}}^{\mathbb{A}})\}
\end{aligned}
$$

where R_m^p denotes p-th resource of the m-th domain. S_{mn}^p and C_{mn}^p denote n-th specification, and constraint respectively of the p-th resource belonging to the m-th domain, where $\mathcal{A}$ to $\mathcal{N}$ are the number of constraints in R_1^1 to $R_{\mathbb{A}}^1$ respectively. Such a formalization maintains the uniqueness of each domain, resource, constraint and its corresponding resource cost.

3.1.3. Macro Operators

The output of the Collections stage is referred to as the Macro Operators (MacOps) data structure, which is a set of candidate cloud resources that could satisfy data-intensive application requirements. Depending on the resource specifications, and constraints, MacOps are constructed from a mapping of a feature in ARI to one or many resources in the corresponding domain $\mathbb{D}$, resource couples in RSpace and are represented as:

$$
f : (F_l, P_l) \mapsto (D_k, R_j^k) \tag{1}
$$

where $l \in \mathbb{I}$, $k \in \mathbb{D}$, and $j \in \{\bar{\mathbb{A}}|\bar{\mathbb{B}}|\cdots|\bar{\mathbb{N}}\}$. Here $\bar{\mathbb{A}}$ denotes the set with $\mathbb{A}$ resources. Equation (1) formulates the process of the ARI features translating into domains and the corresponding ARI preconditions along with RSpace resource specifications and constraints being mapped into one or multiple resources in that domain, forming one or multiple *(Domain, Resource)* pairs. One limiting criteria for such a mapping is that the total number of l's should be equal to the total number of k's, i.e., each feature in an ARI maps to one and only one domain in RSpace. However, no such restrictions apply for a number of resources within the domain. The corresponding ARI preconditions are added to the domain-resource pairs forming MacOps as $(Domain, Resource, Precondition, Cost)$ 4-tuples. A MacOps representation corresponds to an ARI with $Feature$

and *Precondition* parameters. As stated earlier, a single (*Feature, Precondition*) tuple in ARI can lead to multiple *Resources* in MacOps belonging to the same domain.

3.2. Composition

Initially, the *(Resource, Precondition)* pair of the generated MacOps needs to be compared with existing custom templates configurations residing in the catalog. However, if there is no match, a new custom template needs to be created considering user PAC preferences (performance, agility, cost) and using all possible combinations of MacOps (a set of candidate cloud resources that could satisfy data intensive application requirements). This new template will then be stored in a knowledgebase. A Multi Criteria Decision Making algorithm is used in composing a solution in order to create a template or to select the matching templates from the knowledgebase considering user preferences in the form of PAC factors. This process is done through our recommender scheme that chooses the best of the candidate solutions. The different structures presented in the composition stage are:

3.2.1. Custom Template

One or many custom template configurations can be created for a given application depending on the corresponding MacOps results. Each custom template is a combination of 5-tuple (*Domain, Resource, Specification Constraint, Precondition and cost*). Figure. 8 shows an excerpt of the custom template YAML file for a real use case application (WheelSim) that contains a sample description of cloud resources and their configurations. Each template is also associated with 4 different kinds of meta-data. The four meta-data of a template are: (i) requirement to which the custom template is recommended, (ii) PAC factors given by the user for a specific application requirement, (iii) software components that need to be installed, and (iv) resource utilization data to be collected in a deployment. Such a meta-data set can information a CSP on the popularity of certain templates, which in turn can guide the design of our recommenders.

```
tosca_definitions_version: tosca_simple_yaml_1_0_0

description: Template for deploying a HTC Pegasus cluster with
             one master and two worker nodes

topology_template:
    node_templates:
        my_master_server:
            type: tosca.nodes.Compute
            capabilities:
                # Host container properties
                host:
                    properties:
                        num_cpus: 4
                        disk_size: 20 GB
                        mem_size: 8 GB
                os:
                    properties:
                        # host operating system images properties
                        architecture: x86_64
                        type: linux
                        distribution: rhel
                        version: 6.5

        my_worker_01_server:
            type: tosca.nodes.Compute
            capabilities:
                # Host container properties
                host:
                    properties:
                        num_cpus: 4
                        disk_size: 20 GB
                        mem_size: 8 GB
                os:
                    properties:
                        # host operating system images properties
                        architecture: x86_64
                        type: linux
                        distribution: rhel
                        version: 6.5

        my_worker_02_server:
            type: tosca.nodes.Compute
            capabilities:
                # Host container properties
                host:
                    properties:
                        num_cpus: 4
                        disk_size: 20 GB
                        mem_size: 8 GB
                os:
                    properties:
                        # host operating system images properties
                        architecture: x86_64
                        type: linux
                        distribution: rhel
                        version: 6.5
```

Figure 4: TOSCA compliant Custom template YAML configuration excerpt describing the resource requirements for the WheelSim application

3.2.2. Catalog

As the new templates are created they are stored in a catalog for future use. If for a user requirement there exists a template in the catalog which satisfies the application requirement, then it might be chosen by our recommender scheme as a candidate. Initially, the catalog will have few templates generated for the different application requirements. However, over the time the catalog will have critical mass with new custom templates upon deployment of new resources.

3.2.3. Policy Database

Policy database or Rule base, is nothing but the pre-defined set of rules created by the domain expert which drive the questionnaire. The rules are defined for both generation of next set of questions after an initial set of questions and as well as to prepopulate the questions in the questionnaire so as to help novice user. These rules initially are created based on the previous deployment of diverse data-intensive applications that required heterogeneous resources, as well as logical rules that guarantee compatibility and functionality of cloud resources.

3.3. Consumption

The last stage of the custom template life cycle focuses on the resource deployment. Once the custom template configurations are generated or retrieved from the catalog for a particular application's requirement, the configurations (along with the cost) are presented to the end user. Three different templates are selected for the user ('green', 'red' and 'gold'). We consider the closest solution to the user requirement as the 'green template', the template that presents lower cost as the 'red template' and the template with more resources for a certain feature (i.e., memory) based on user preference as the 'gold template'. This preference is explicitly selected by the user during the KIS interactions.

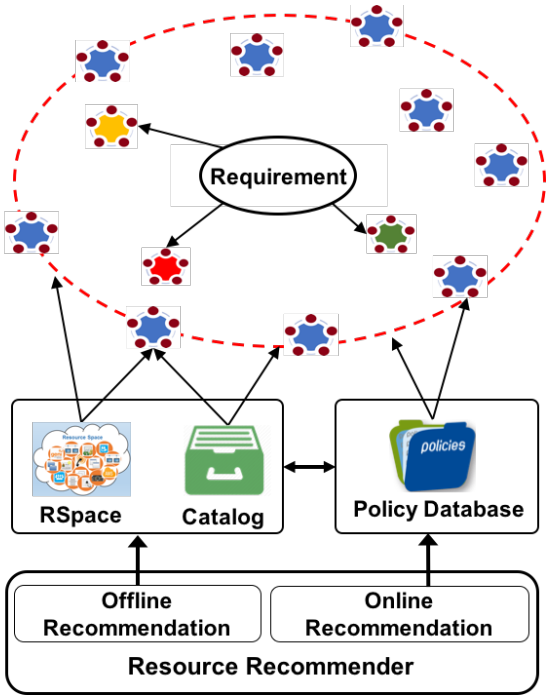


Figure 5: KNN algorithm to recommend templates from Catalog or compose a solution from RSpace guided by policy database.

Once the user chooses a particular template, corresponding infrastructure resources are automatically deployed and credentials for infrastructure interaction are available to the user. The deployment process utilizes the APIs for different CSPs and Docker container technologies such as 'Docker Hub' for software deployment. Once the infrastructure is deployed, a monitoring manager verifies the deployment and then collects the information about user account creation, IP configuration, user privileges granted, customized software and availability and network connectivity. This data along with the application specification is stored and is available as part of the template meta-data within the catalog. For this work, we use a simple cost per time-unit model and investigations on a more sophisticated cost model are beyond the scope of this paper.

4. Recommendation Scheme

4.1. Overview

Our middleware considers two different levels of recommendations: *offline initial recommendation* and *online iterative recommendation*. First, the *offline initial recommendation* is enabled in the beginning when the user deploys a new science gateway application setup on a cloud platform. The user interacts with the KIS to provide the application requirements including PAC factor preferences as explained in the previous section. Once the requirements are confirmed by the user, the recommender matches those requirements with the Catalog or RSpace to provide initial (solution) templates. This first level primarily depends on the user response to the questionnaire.

The second *online iterative recommendation* level is enabled after the application is deployed and is initiated when it shows signs of sub par performance relating to application demands. Most likely in such cases, the application requires different quantity and type of cloud resources from the initial request as the resources are not suited as initially expected. For example, variation in the application behavior leads to significant modification in a particular cloud resource utilization. Hence, this triggers notable changes based on data that has been collected through resource monitoring. Once the data is collected and analyzed the recommender will suggest possible changes to *adapt* the current architecture or will suggest a *migration* to a different CSP in order to satisfy user requirements. A graphic representation of our offline, online recommendation is shown in Figure 5, and details of our offline and online recommenders are presented in the following subsections.

4.2. Offline Initial Recommendation

In this section, we present details of our template recommendation algorithm that finds the most closely matching templates for a given user's requirement. After considering different existing classification algorithms for template classification and matching prediction, we selected the popular KNN scheme due to its simplistic as well as effective nature for our purposes [42]. Our algorithm extends the k-Nearest Neighbors (KNN) algorithm to find the three most closely related templates (i.e., 'green', 'red' and 'gold') in the catalog as shown in the Algorithm 1. Preferred dimension (pDimension) input is explicitly defined by users through the KIS and it is used to prioritize resource requirement during template selection process. From the MacOps, we construct a requirement vector V (reqVector) having (r_1,r_2,r_3,....,r_n) where r_i is the pre-condition for each resource in network, storage and computation domain.

The recommender system uses three different knowledge-base as shown in Figure 5. They are listed as *RSpace* - which is available pool of resources, *Catalog* - the templates stored in repository and *Policy Database* - which has rules to resolve conflicts or ties when there are two similar templates available for a requirements. Using the requirement vector constructed from the MacOps, KNN tries to find templates from the catalog. The algorithm first, calculates the distance between reqVector and templates in catalog and fetches the templates with distance less than the threshold. Following this, the reqVector is compared with n-dimensional vector with R_i components $\in$ RSpace where R_i contains: No. of Cores, RAM, Storage and Bandwidth, and so on. Then, the distance between R_i and reqVector is calculated using the formula 3 for $R_i \in$ RSpace.

$$D_i = w * \sum_{j=1}^{j=n} V_j * abs(R_{i,j} - V_j) + w * V_4 * max(0, R_{ij} - V_j). \quad (2)$$

The weights in the above equation are determined by collecting preferences from the user and by using the Analytical Hierarchy Process described in Section 2. The weight factor w is used to scale the distance. Note that multiplying each term by V gives more weight to the distance in the dimension which

Algorithm 1: Offline template recommendation

```
function Decision(MacOp[1..m], catalog[1..n], pDimension, reqVector)
    //set the threshold distance
    D ← 2;
    //filter templates from catalog whose distance from reqVector is less than threshold
    //calculate templates distance to reqVector
    foreach t in f do
        tList ← distance(t, reqVector, pDimension);
    end
    //filter k number of candidate templates
    candidateList ← nearestNeighbors(D, tList, reqVector);
    if candidateList is not empty then
        //AHP Multi-Criteria decision making algorithm to choose - RED, GOLD and GREEN from the candidateList
        //select template with cost as objective
        RED ← ahp(candidateList, cost);
        //Select template with both cost and performance as objective
        GOLD ← ahp(candidateList, cost);
        //select template with performance as objective
        GREEN ← ahp(candidateList, performance);
        return RED, GOLD, GREEN else
            //Generate possible candidates combination of MacOp
            newTemplates ← MacOpComb(MacOp);
            //AHP algorithm to choose RED, GOLD and GREEN from generated templates
            RED ← ahp(newTemplates, cost);
            GOLD ← ahp(newTemplates, cost, performance);
            GREEN ← ahp(newTemplates, performance);
            //Add these newly created templates to catalog
            catalog ← catalog + (RED+GOLD+GREEN);
            return RED, GOLD, GREEN
        end
    end
end
```

Algorithm 2: Online resource adaptation

```
function Adaptation(tInfo, monitoringData, policyDatabase)
    //Adaptation algorithm is given current template info, monitoring data and user defined scaling rules if any
    //Check if user defined scaling rules are present
    if tInfo.scalingRules == True then
        //Read user defined threshold from template info
        uPerf ← tInfo.upperPerformanceThreshold;
        uTimeLimit ← tInfo.upperTimeThreshold;
    end
    while True do
        //Check if current performance is more than the threshold
        if (currentPerf > uPerf) and (cumulativeTime > uTimeLimit) then
            if tInfo.scalingRules == True then
                //Case a: add or remove VM instances based on the rules
                addOrRemoveResource(newTInfo);
            end
            if isScalingUpperLimit == True then
                //Case b: change the VM type based on the policy database changeVMType(tInfo, policyDatabase);
            end
            if isMigrationPoint == Ture then
                //Case c: get new reocmmendation Obtain new template from the Decision Engine. The Decision Engine will get user confirmation
                newTInfo ← getNewRecommendation(tInfo, policyDatabase);
                //backup data and app from the old template
                bkpDataAndApp(tInfo);
                //migrate the application to new template
                migrate(tInfo, newTInfo);
                //kill old resources
                deprovision(tInfo);
            end
            //update the scalingUpperLimit flag, set it to true if maximum number of VMs has been added
            update(scalingUpperLimit);
            //update the migrationPoint flag, set it to true if change in VM type is done more than 3 times and still no improvement in the performance
            update(migrationPoint);
            //Update user performance and thresholds
            uPerf ← newTInfo.upperPerformanceThreshold;
            uTimeLimit ← newTInfo.upperTimeThreshold;
        end
    end
end
```

is rated highest by the user. This dimension is further referred to as the Preferred Dimension.

To streamline the process of matching the user preferences with the appropriate templates, we are classifying them into RED, GOLD and GREEN using a simple multi-criteria decision making algorithm. For performance as objective function, users can select the GREEN option, RED for cost, and GOLD for both performance and cost. If there are no templates in the catalog, then the algorithm composes a new solution. It again uses KNN with reqVector and RSpace to select different resources from the available resource pool. The fetched resources are again fed to a multi-criteria decision making algorithm to compose three templates i.e., RED, GOLD and GREEN. Finally, these will be stored in catalog and are presented as new recommendation options to the user.

4.3. Online Iterative Recommendation

We allow provisioned federated resources to be adapted and refined automatically after the previous phase using a novel online recommendation scheme shown in the Algorithm 2. After the initial recommendation and deployment, the monitoring engine starts collecting resource utilization data of all the resources in the deployed template. The online recommendation can force the system to operate with the existing template VM configuration or suggest modifications based on fine-grained or coarse-grained considerations.

There are three possible scenarios under consideration in our approach. *First scenario: fine-grained control*, if there is no past history of scaling or (performance) symptoms of slight deviation from the acceptable user preference after scaling, then the decision engine will send the information to the deployment engine to change the number of VMs as fine-grained modification. *Second scenario: semi-coarse-grained control*, if there was past VM scaling history and yet the performance of the application continues to remain unacceptable, the decision engine will suggest to the user to change the VM type as semi-coarse-grained modification. If user accepts the newly recommended solution, then the decision engine again contacts the deployment manager to implement them accordingly.

We remark that the above two scenarios correspond to 'flex points' that are short-term in nature as considered similarly in [28]. The resource adaptation is influenced by application-specific adaptation rules for 'scale-up/down' or 'scale out/in' of cloud resources in a manner that is compliant with TOSCA standards. *Third scenario: coarse-grained control*, if there are major performance gaps in spite of the past history of modifications from the second scenario, then the adaptation triggers coarse-grained modifications. In this case, the necessary adaptation corresponds to a 'rebuild point' that is part of a long-term adaption plan, where we suppose that the adaptation will require a completely

new custom template with a drastically new cloud architecture to meet application requirements. Correspondingly, the user is encouraged to interact with the KIS questionnaire with a clean-slate set of requirements, and the recommendation engine searches for a new custom template in the catalog that has relatively much higher capacity resources with a different cloud architecture. Upon user approval, the recommendation engine contacts the migration engine and with the help of deployment engine, it automates the reserving new resources and application setup transfer from the previous template. This above described process is iterated till the performance is met within satisfactory limit simultaneously accompanied by systematic close down of prior allocated resources.

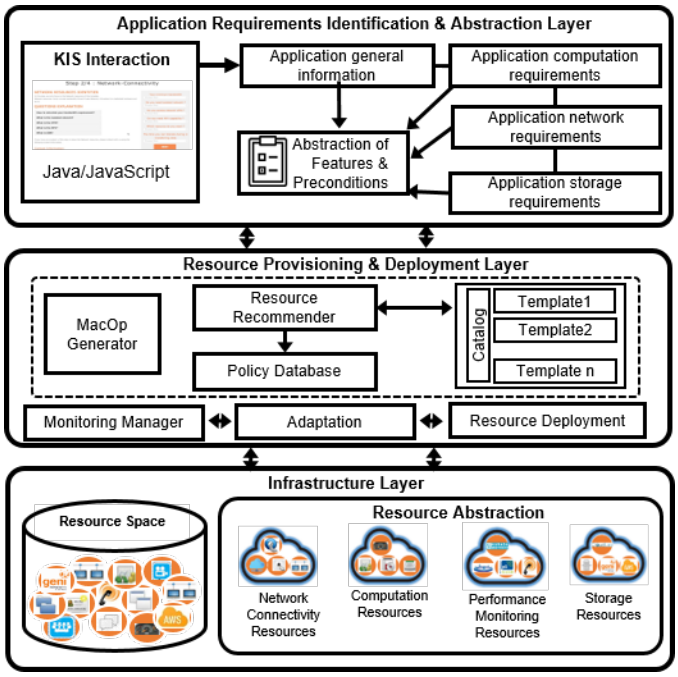


Figure 6: System architecture diagram for application requirements abstraction and hybrid cloud resource deployment using custom template middleware.

5. Application Workflow Integration

5.1. Application Architecture and Implementation Design

The middleware is developed using Java Struts 2 web framework. For the front-end user interface, JSP/JavaScript and JQuery are used and entire application logic is written in Java. We use the MySQL relational database to store the abstracted cloud model (RSpace). From an architectural point of view, provisioning cloud services involves identifying cloud resources required for the user requirements, deploying infrastructure in the selected CSP and installing and configuring any software that user needs in the deployed federated/distributed resources. The last step i.e installing and configuring software is done by using Docker container that allows software deployment on a variety of platforms without being constrained to software dependencies. We have made the portal code and recommender scripts along with data structures, RSpace data and GENI configurations openly available at [40].

Figure 6 shows the middleware architecture diagram (divided into three layers) to automate application requirements abstraction and federated cloud resources deployment. In the 'Application Requirement & Abstraction Layer', users interact with the middleware through the KIS user interface to identify the application requirements. This layer also generates ARI from the user input captured by the KIS and it is passed to the 'Resource Provisioning & Deployment Layer' where the ARI is compared with the RSpace resulting in the creation of MacOps. The MacOps are used by the 'Resource Recommender' module to create/reuse custom templates and catalog these templates. Once a user selects a particular custom template, resources will be automatically provisioned through the 'Resource Deployment Module'. The 'Infrastructure Layer' abstracts the resources from different CSPs and calls the correct API to interact with a specific CSP. Infrastructure Layer receives the template to be deployed from resource provisioning and deployment layer and calls the relevant APIs to allocate resources on corresponding CSP infrastructure. The 'Monitoring Service' verifies that the resources are deployed successfully. All the inter-process communication between different layers is implemented via RESTful web services.

Our 'Resource Deployment Module' integration with different cloud providers occurs at two levels. A user-selected custom template is presented to the 'Resource Deployment Module' in a TOSCA compliant format. This module then extracts relevant implementation information from the template (such as e.g., the cloud service provider, type of the resources and its specifications) and interacts with the infrastructure layer components via their APIs. Given that our cloud template recommendation considers heterogeneous resources, we needed to have our 'Resource Deployment Module' to be compatible with multiple cloud providers. In order to add a new cloud service provider to our middleware, three different kinds of information needs to be added in our knowledgebase. First, we have to update the *RSpace* data structure, which has information about all available cloud resources from the cloud service provider that are relevant to an application community. Secondly, we update the policy database with pricing information of various resources and rules that are essential for the composition step of the custom template lifecycle (shown earlier in Figure 2). Thirdly, we update our 'Resource Deployment Module' to use the cloud provider's specific APIs to deploy and manage resources once a custom template solution is chosen by a user for deployment in the consumption step of the custom template lifecycle.

A major challenge in our implementation for using multiple cloud providers is to be up-to-date with the dynamism in the cloud provider APIs and the inherent sustainability issues handling legacy features. Owing to our use of the catalog knowledgebase, expert users successfully creating custom templates will foster novice users to adopt more latest features/capabilities offered by cloud providers. Specifically, in our implementation we use Amazon AWS Java cloud API and the *genilib* python library for GENI, respectively as part of the cloud resource deployment that are frequently used by our application community (i.e., advanced manufacturing, bioinformatics). Addition-

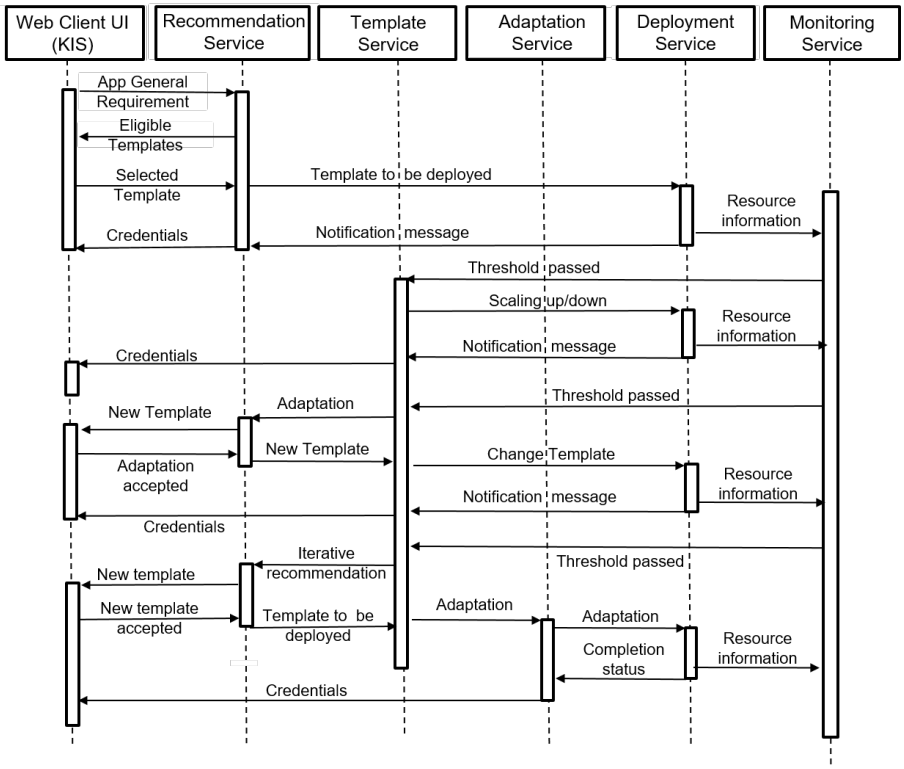


Figure 7: Sequence diagram of all the steps involved in collection of requirements, composition & recommendation, deployment and adaptation/migration of a custom template solution.

ally, it is possible to explore integration opportunities of generalized cloud API libraries such as *libcloud*, *jcloud*, and *deltacloud* for deploying heterogeneous cloud resources as mentioned in Section 2 (Related Work). However, these open-source libraries face the same challenges in being up-to-date with the dynamism in the cloud provider APIs and the inherent sustainability issues handling legacy features. Fortunately, our approach is to be TOSCA compliant, which makes our implementation agnostic to cloud infrastructure changes. TOSCA compliance allows us to have simpler application deployment in popular cloud platforms that are TOSCA-compliant, and enables multi-cloud deployments without any specific cloud provider lock-in.

5.2. Sequence Diagram for Component Interactions

The sequence diagram in Figure 7 shows the chronological events in the life cycle of a custom template as shown in Figure 2. It begins with user interaction with KIS in order to specify the application requirements to the recommendation system. This, in response, presents the user with three templates ('green', 'red' and 'gold') from the template service according to cost, agility and performance preferences. These event sequences correspond to the offline initial recommendation phase. Once the user submits the template information to the KIS, the control comes back to the recommendation service. It supplies this user-specified template configuration information to the Deployment Service for implementing the user resource requirements. The deployment service reserves the resources on the selected cloud, and completes the install and configuration steps for the new architecture with the necessary softwares. The control for these newly allocated resources is now transferred to the monitoring service with regular updates to the user.

After the deployment, if there are scaling rules defined by the user then these rules trigger adaptation service. The Adaptation Service can take three different actions and correspond to the online iterative recommendation explained in Algorithm 2.

The adaptation service ensures proper shut down and resets previous resources from the prior allocated template. Finally, the access credentials of the new resources are given to the user. In all of the above cases, the monitoring service starts to collect data with any kind of modification in the existing template. This monitoring information is regularly updated to the adaptation service to ensure stable performance that meets the user PAC preferences.

The design of our initial KIS questionnaire and implementation of our middleware is based on our experience from working on real-world use cases with bioinformatics community users (e.g., SoyKB) [41] as well as advanced manufacturing community users (e.g., TotalSim) [43]. As noted earlier in Section 3.1, it is important for the KIS design to have a deep understanding of the underlying application community knowledge, and requires close collaboration with the application community. We worked closely with several community domain experts (both novice and expert users) and studied cloud provider offerings to input pertinent rules into our policy database, and populate an initial set of custom templates in our catalog for use with our KNN algorithm. Any implementation that uses a

catalog faces a "cold start" issue, where early-on in the catalog creation, no solutions exist but over time many successfully deployed solutions with corresponding resource benchmark information (i.e., application utility functions) are populated. Fortunately, there already exists knowledge of the popular workflow requirements and templates of application-specific configurations in application communities via archived publications and online documents. Hence, we were able to use a combination of already published literature and discussions with application community collaborators to populate the catalog with initial templates of application workflow deployment architectures involving: (a) use of workspaces featuring virtual desktops that host scientific software such as Paraview or Matlab, (b) web services based packages to query databases to transfer/analyze distributed data sets, (c) access to cloud resources using application community-specific batch systems that are widely used (e.g., SoyKB uses HTCondor/Pegasus [43]; TotalSim uses SGE/OpenFOAM [41]).

6. Performance Evaluation

Implementing our custom template middleware can provide seamless access to the users (novice and expert) to federated cloud resources configured based on the requirements of the data-intensive application. Particularly, users without technical or cloud platform experience can easily interact with context-aware questionnaire (KIS) and take advantage of our recommendation scheme to provision the pertinent resources. Additionally, upon automated deployment of cloud resources, credentials will be sent to the users along with access instructions. This whole process reduces the resource provisioning time drastically and removes guess work in cloud resource allocation complexity, arising from manual approaches. In addition, users can save their previous successful solutions as custom templates in the catalog for portability and future purposes.

We are evaluating the middleware for custom template composition for resource allocation in four main steps. It begins with the case study of private cloud architecture and a corresponding cost-performance analysis. This is done to transform a manufacturing industry simulation from a private cloud platform to a public cloud platform. In the next part, we demonstrate the benefits of the novel KIS with the use of a basic knowledgebase about resource capabilities. The third set of results evaluate the recommendations for various grades of cost-performance user requirements ('green', 'red' and 'gold'). Finally, we quantify the accuracy of the questionnaire's response in meeting (both novice and expert) user requirements with regards to identifying the ideal template.

6.1. Data-intensive Science Gateway Application Case Study

In this subsection, we present the effectiveness of our middleware implementation in the form of a case study for an advanced manufacturing of data-intensive science gateway application [43]. Our case study pertains to a small-business advanced manufacturing company viz., TotalSim (located in Dublin, OH) who integrated our custom template middleware solution and provided their cloud resource requirements and pent-up business development needs as input to the KIS. A model requirement input is presented below:

- Highly available cluster infrastructure resources required with support of batch processing execution
- Nodes require 2 cores with 4-8 GB RAM and 20 - 40 GB storage and bandwidth of 10-15 Mbps
- Require high-throughput computing software framework such as HTCondor available upon resource provisioning
- Require customized queuing software available on master node upon resource provisioning (stored in Github repository)
- Require binary files available on new cluster environment (sources in Github repository)
- User Preference input for pDimension: The preference feature selected is RAM Memory

Based on the requirements of this application collected using the rule based questionnaire approach shown in Figure 8, we will analyze the trade-offs between cost, performance and agility w.r.t. to the private and public cloud platforms. The motivation for workflow transformation from private resources to a public cloud, as listed in Table 1, are well-established. However, we demonstrate that the addition of our middleware can help private cloud resources to deliver automatic autoscaling of compute resources in cost-effective manner.

6.1.1. Private Architecture

For initial exploration, application characterization for CPU and RAM are presented in Figure 9. As compared to increase in virtual cores, there is much significant drop in the execution time with the availability of larger RAM size as shown in Figure 9 (a), which allows us to conclude that it is a memory-intensive application. Obviously, the maximum performance gain is observed on cumulative increase of both memory and compute resources i.e., from 1 vCPU and 2 GB of RAM to 8 vCPUs and 32 GB of RAM. Figure 9 (b) shows that initially the timing (y-axis) reduces by increasing the number of cores. Although PR has 150 cores, the best performance is obtained by using only 96 cores. Any addition in the core count beyond 96 cores causes the "law of diminishing returns" to become dominant and the overheads increase due to the rise in the interprocess communication costs. Hence, it is important to scale the resources by matching them with the application requirements. To correlate, similar execution time and cost scalability characterization for different workloads for the public cloud (AWS) are shown in Figure 9 (c). The AWS resources were deployed using StarCluster, which is an open source cluster-computing toolkit for Amazon Elastic Compute Cloud service. We use this service in order to automate and simplify the process of building, configuring, and managing clusters of virtual machines on Amazon EC2, suited for distributed and parallel computing applications and systems.

Table 1: Motivations for transformation of a workflow from a private resource to a public cloud

Task	Private Resources	Amazon AWS Resources
Collaboration	Asynchronous i.e., using email, screenshots	Synchronous i.e., WebEx on virtual desktop, cross-platform/device operation
Data storage & sharing	Mailing DVDs and copying files with `scp` utility	Cloud storage - Dedicated VM with NFS volume attached to it
Network for data transfer	Unreliable public Internet connection	Amazon internal fast network for data transfer among the VMs and public Internet for external transfer
Resource procurement	Local cluster with limited resources	Infinite number of EC2 VMs that can handle massive demand bursts
Pricing analysis	Low cost but limited resources	Medium/high cost but unlimited resources

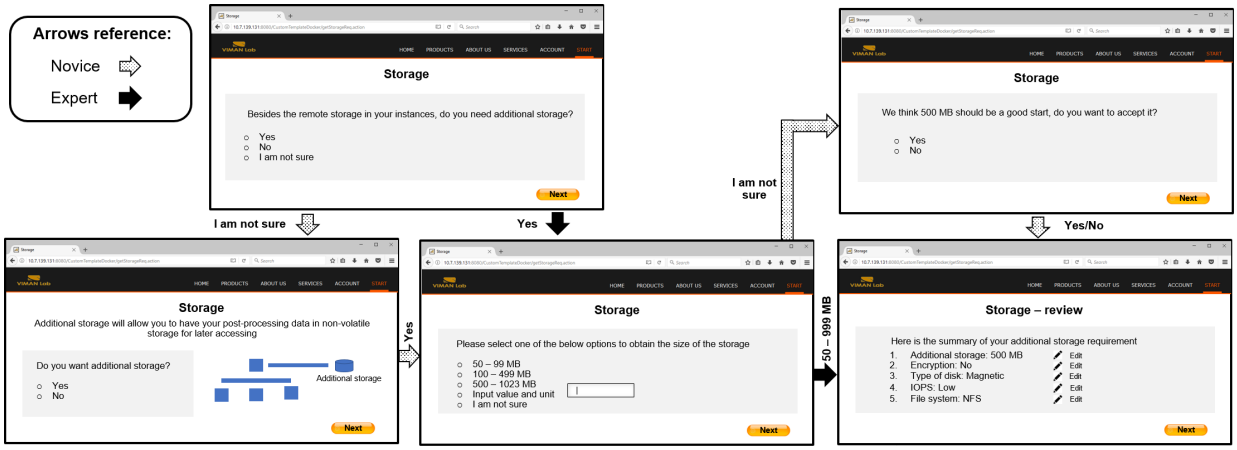
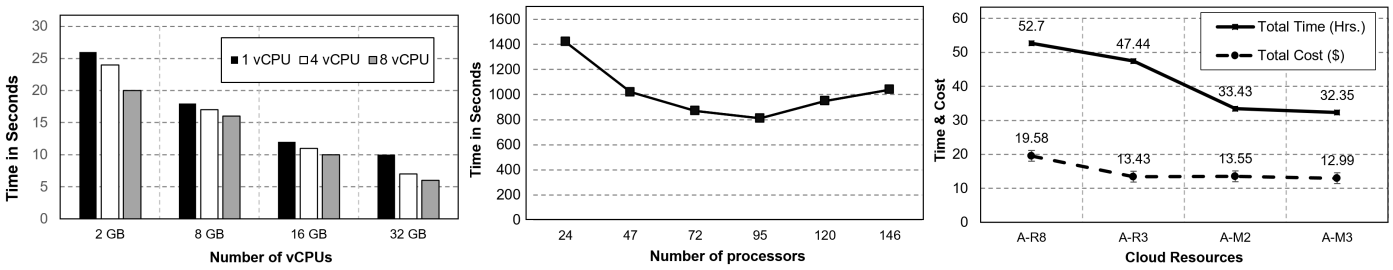


Figure 8: KIS is a rule based web questionnaire that collects general data-intensive application information, network-connectivity, storage, computation resources and software requirements. Based on users interaction with the KIS, an internal module (policy database) pre-populates fields with pertinent values to help users decide whenever they are not sure about their inputs.



(a) (b) (c)

Figure 9: (a) Performance results with varying vCPUs and RAM for the private cloud, (b) Correlation between the runtime of the workflow and the number of CPUs for the private cloud, (c) Corresponding performance and cost results for the public cloud. The initial recommended template computed the WheelSim workflow in 52.7 hours with a cost of $19.58. However, once the monitoring engine detected an over-provisioning of resources, the middleware recommended a second template that computed the workflow in 47.44 hours with a cost of $13.43. The recommendation process continues until it minimized the computation time and cost.

6.1.2. Rule Based Questionnaire Implementation

The focus of the experiments in our prior work [8] was on a single VM deployment as we considered a basic questionnaire for novice and expert user. In this paper, we are extending the work scope by recommending an overall solution template with multiple cloud resource requirements. Thus, we do not just limit our recommendations to a single or isolated cloud resource, but also consider heterogeneous resources from multiple cloud infrastructures that are potentially suited for our data-intensive science gateway application requirements. In order to achieve this objective, the questionnaire is updated with supplementary steps to elicit additional information on the architecture, topology, and other considerations. Details about these related parameters are specified in Table 2. We have compiled this list of information based on our previous work with the advanced manufacturing application collaborator [43].

It was not straightforward process to make the requirement elicitation process user-friendly, complete and consistent, as the new questions have imparted complexity to the original questionnaire. In that direction, our main idea is to be able to predict the best possible choice for the infrastructure requirements as the user goes through a series of questions with typically binary answers (yes/no) for requirement elicitation about their application. The answers to these enquiries are used to populate the parameters which can be translated to features which are in-turn used to make the infrastructure decisions. Our rule based context-aware method allows us to efficiently interact with the users adjusting the inquiry flow according to user's comprehension level of cloud infrastructure and application requirement. In other words, users with performance-optimization goals will

Table 2: Comprehensive view of Novice vs Expert User Requirement for Reserving Custom Template

Template Parameters	Exact Information	Partial Information	Minimalistic Information
RAM (Memory)	10 GB	6 GB	6 GB
CPU Cores	2		
CPU Frequency	2 GHz		
OS (Type, Platform, Version)	Unix-type, CentOS 6	Unix-type, CentOS	Unix-type
Linux Server OS availability	CentOS	CentOS	
Windows Server OS availability	No	No	
Storage	50 GB	30 GB	
Number of VM Instances	1	1	
Required network topology	No	No	
Application Type	Memory-intensive	Memory-intensive	Memory-intensive
Physical location of resource	Ohio	Ohio	
Type of connectivity to remote resources	Layer 2		
Network bandwidth Mbps (within the data center)	100 Mbps		
Network bandwidth Mbps (from data center to private resources)		50 Mbps	
Type of resources	Virtual Desktop	Virtual Desktop	Virtual Desktop
Type of cost per instance	per execution	per execution	
Type of cost per storage	per GB	per GB	
Capacity of master node different from slaves?	No	No	No
If different master node specs better than slaves?	No	No	No
GPU Requirement	Yes	Yes	Yes

go through a separate set of questions, as compared to price-concerned application-owner. As the number of interactions increases and the knowledgebase maturity grows, the questions will increase in specificity and cover advanced concepts. Expert users interacting with our context-aware questionnaire are expected to supply closer to exact requirements and hence get better (custom template) recommendations as compared to the novice users. It can be foreseen that the novice user will have lower engagement with the questionnaire and so the recommender will suggest suitable templates with minimal interactions with the same set of questions.

6.1.3. Cloud Architecture Transformation

To analyze the private to public cloud transformation we compare their price versus performance metrics. We began with publicly available CSP resources data such as Amazon resources: m3.2xlarge (A-M3), m2.4xlarge (A-M2), r3.4xlarge (A-R3) and r3.8xlarge (A-R8), and private resources used by the Manufacturing company such as: Ruby (PR-R), Oakley (PR-O) and Glenn (PR-G). Figure 9 (c) shows the results of our middleware implementation. It is divided in two sections, for private architecture and AWS resources, sorted by execution time and cost. The reason of having any resources included as a potential solution will depend on the user priority established for the particular application. A detail explanation is shown in Figure 10, where the same type of resources present different ranking depending on the preference such as: Resources availability, Computation time and Cost.

Availability metric represents a numerical score of the user's ability to reserve the suited resource (diversity and amount) as necessitated for an application during a specific resource demand period. Multiple applications typically compete for the same resource units in a private cloud, which lowers this score for a PR in comparison with a public cloud. In Figure 10 (a), this score is calculated for the WheelSim application by averaging all the past data corresponding to when the resources are available for user allocation. Public cloud has unlimited availability given their elastic resource characteristics to meet user demands. Hence, they are assigned 100% values for the availability calculations.

Some of the cloud resource requirements (RAM, Storage Size and Bandwidth) specified in Section presented range values. Consequently, boundaries are established through vectors (V_{min} and V_{max}) and the reqVector is constructed with V_{mid} vector. Some of the vector values are: V_{min} = (2, 4, 20, 10), V_{mid} = (2, 6, 30, 12.5) and V_{max} = (2, 8, 40, 15). we use the evaluation metric, *Resource Demand* in this set of experiments. This metric is defined as the amount of resources needed for different templates.

Our recommender scheme matches this requirement with different CSP capabilities and presents the available resources to the user for science gateway application deployment. Once the user selects one of the proposed solutions (i.e., among the 'green', 'gold' and 'red' options), the infrastructure is deployed and pre-configured with all the necessary software using our Deployment Engine that uses technologies such as Docker. Credentials along with instructions to access the new infrastructure are made available for the user. Once this process is done, the solution template is stored in the catalog for future reuse.

In order to demonstrate the effectiveness of the pDimension variable, we perform two experiments using the advanced manufacturing application implementation. In the first experiment, all resources have the same priority, hence a pDimension was not defined. However, for the second experiment, 'RAM' was selected for pDimension. From Figure. 11 (a), we can observe that the pDimension variable affects the output results because the candidate templates differ in the number of recommended resources (i.e., in the Resource Demand). Results in Figure. 11 (b) show that the cloud resources are out of the boundaries. However, results in Figure. 11 (a) show candidate templates are found to be within the upper and lower boundaries as well as those that are close to the upper boundary. Majority of our experiments reveal an interesting phenomenon related to the cost, where the 'green' template generally incurs high cost and the associated vector is close to the upper bound. Moreover, the 'gold' template (template with the highest values based on

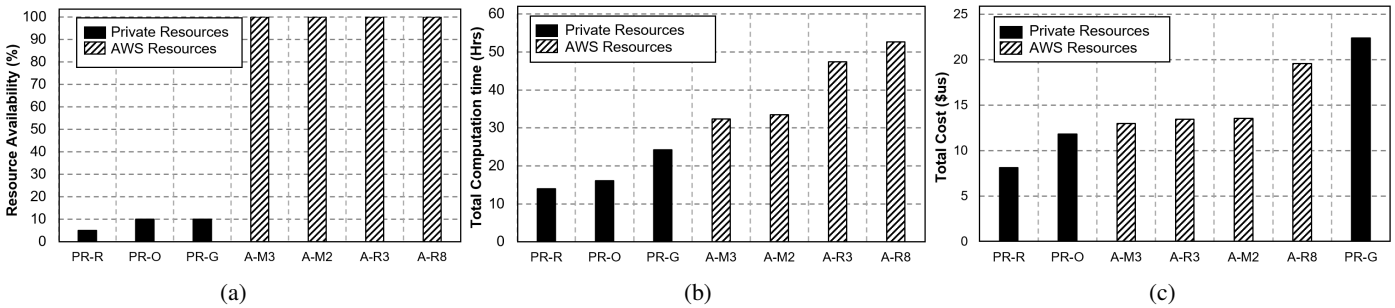


Figure 10: We executed the WheelSim application over 7 different instances types from 2 different cloud resources (private resources (PR) and public cloud resources (AWS)). Values in X-axis correspond to the abbreviation of the type of instance utilized for the experiments. Generated data is plotted based on three parameters: (a) Availability, where PR present limited availability over AWS, (b) Computation time, that takes into the account the time needed to transfer processed data from remote to private location, where private resources present lower computation time and AWS higher computation time, transfer time for a 15 GB file in the case of PR-R, PR-O, PR-G, A-M3, A-M2, A-R3 and A-R8 is 25, 25, 25, 53, 53, 53, 40 minutes, respectively, (c) Total cost, where some private resources i.e., GENI with PR-R and PR-O present a lower cost in comparison to using AWS.

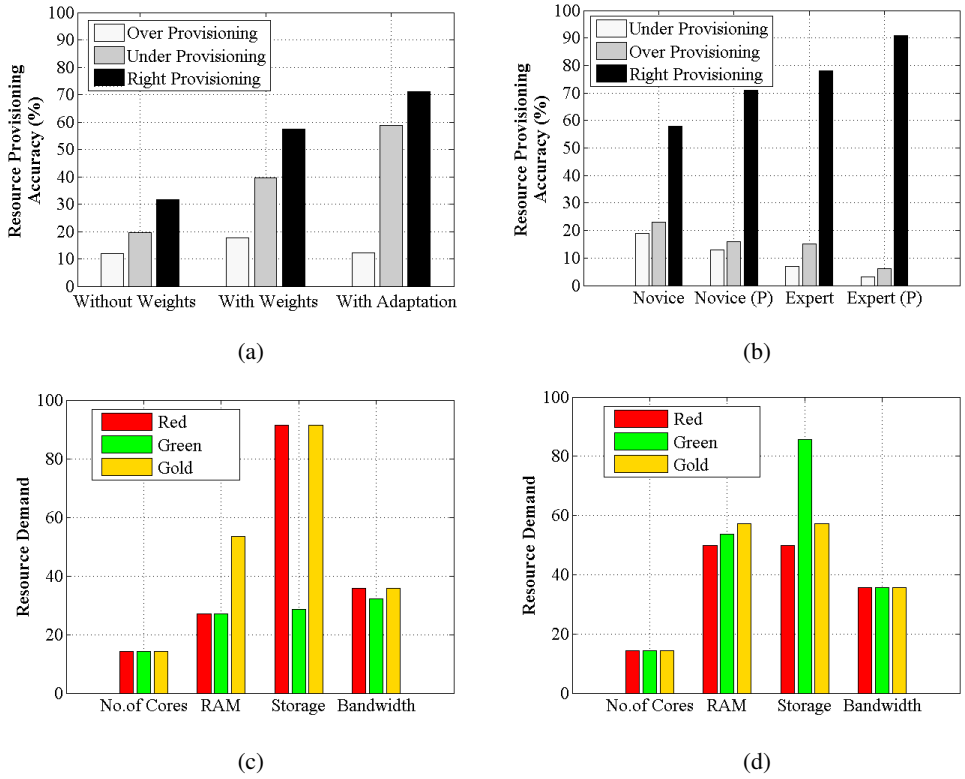


Figure 11: Accuracy for recommended results: (a) This figure presents results based on Figures. 5 and 6 of work done in [26], where the recommender schema present a maximum accuracy value of 71% ; (b) Present our middleware results based on four different input data: *Novice* that represents Requirements Without Preference), *Novice (P)* that represents: Requirements With Preference, *Expert* that represents: Requirements Without Preference and *Expert (P)* that represents Requirements With Preference. The maximum accuracy obtained is 92%. The reqVector presents the following requirements: 2 Cores, RAM memory between 4 and 8, Storage capability between 20 and 40 GB and bandwidth between 10 and 15 Mbps. Gold, green and red templates are presented as candidate solution with normalized values. (c) pDimension is not specified hence some templates are out of the boundaries; (d) pDimension for *RAM Memory* resource is specified, hence all templates tend to be close to the *RAM* upper bound, and all templates are inside the ranges.

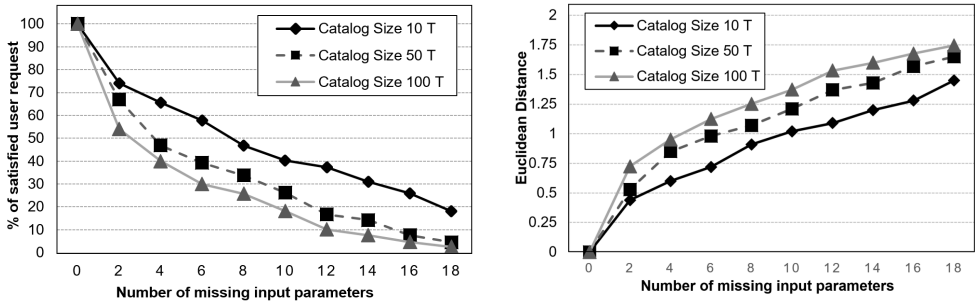
RAM pDimension) incurs lower cost in comparison with the 'green' and higher cost than the 'red'. Finally, the 'red' template that presents the lowest cost, as well as the vector, is close to the lower bound.

6.2. Experiments and Results Discussion

6.2.1. Recommendation System

For our evaluation experiments, we used AWS resources. We setup a testbed with distributed heterogeneous resources that can be discovered and configured by our recommendation scheme using the AWS API capability. To mimic the different data centers, we considered different zones that are distributed across many cities.

To evaluate our middleware utility, we use a *Resource Provision Accuracy metric*, which we define as the similarity of cloud resources presented in templates, versus the similarity of the cloud resources in requirements of data-intensive applications. We evaluate the accuracy of our recommendation sys-



(a) (b)

Figure 12: (a) Percentage of user requests satisfied vs number of missing attributes. The percentage of user receiving recommendations closer to the ideal solution decreases as the number of missing attributes increases. In addition, while keeping the number of requests constant and increasing the number of templates in the catalog, there is a reduction in the percentage of identifying the ideal solution template. (b) Euclidean distance between the ideal solution and the template recommended by varying the number of missing input parameters.

tem by considering resource requirement boundaries created based on reqVector. The boundaries are calculated based on the input ranges selected by the users at the time of choosing cloud resources through the KIS (e.g., required bandwidth range: 10 - 15 Mbps). This information is used to create additional boundary vectors. V_{min} is the lower bound of the requirements, V_{max} is the upper bound of the user requirements and V_{mid} is the vector constructed using the mid-values for the features. We test our recommendation system with generated data for 60 user requirement requests. We implement our recommendation scheme for 30 novice users (users who do not provide enough information to the KIS) and 30 expert users (who completed most or all fields in the KIS). The recommendation scheme is also evaluated by considering users who explicitly specified pDimension variable (resource preference selection) and users who did not specify any preference (all resources have equal importance). Finally, we compare our results with a prior scheme detailed in [26] that also has a recommendation approach as mentioned in Section 2.

Figure 11 (c) shows results obtained with our scheme which clearly presents improved results in comparison with Figure 11 (d). Results in Figure 11 (c) are cataloged in different groups that correspond to novice and expert users. Within those categories there are two sub-categories which represent users who consider all resources with the same importance and users that explicitly set a pDimension to determine preference or priority of some resource component. Results for novice users show that our recommender scheme achieves nearly 58% accuracy when pDimension is not applied and 71% accuracy when pDimension is applied. pDimension represents an improvement of nearly 13%. Similarly, results for expert users show that our recommender scheme achieves nearly 78% accuracy when pDimension is not applied and 92% when pDimension is applied. pDimension represent an improvement of 14%. Overall our recommender scheme accuracy depends on the user input and accuracy is high when the user has some level of cloud knowledge or experience, and accuracy is low otherwise. Also, pDimension effectively targets resources based on the resource priority specified.

6.2.2. Novice versus Expert User Questionnaire Response

In this section, we will evaluate the effectiveness of our recommendation system for a novice versus expert user with the existing information from previous users that is available in form of the knowledgebase. In order to design an automated, consistent and effective requirement elicitation process, it is important to understand how the prediction accuracy changes with the modification in the missing parameters, minimum number of requirements to obtain consistent decision-making and especially how most-influential features will influence several other features. The premise of the problem is that the user (especially the novice type) has a partial knowledge about their application requirements and even lesser hardware/software understanding and how it can be matched to the existing CSP infrastructure capabilities. Even the expert user may not have a complete knowledge about parameters and their correlation with complete picture of cost, agility and performance factors. Finally, we support our design by the evaluation results from simulations over a large number of user requests with different template sizes.

The evaluation of our recommender system is extended by quantifying its ability to provide solutions for the novice user requests with respect to the ideal template. Since the novice user has an incomplete information on the requirements for the application and cloud features, their requests usually have missing responses about the infrastructure parameters in the questionnaire as shown in the Table 2. Hence, it is important to conduct a systematic experimentation to gauge the extent of accuracy errors as the amount of missing values increases. For this purpose, we are simulating the requirement elicitation with 20 input parameters with about hundred user request and randomly modifying the number of missing values and the templates stored in the catalog. The ideal template is predefined as the one that is recommended to a particular user request with all the input parameters correctly supplied to the recommender system. As per the actual scenario, attributes for each user request will be matched with the existing custom templates in the catalog. For the same user request, we are systematically deducting the random number of input parameters to simulate the impact of the missing values and use Euclidean distance with the KNN algorithm to identify the template in the catalog clos-

est to the user request. Obviously, accuracy improves if there are higher number of matches. For a constant number of missing values, we are calculating the error using the equation given below:

$$SUR = \frac{\sum_{i=1}^{Maxiterations=100} \frac{CM_i}{UR_i}}{MaxIteration} x100. \quad (3)$$

where,
SUR = Percentage of Satisfied User Requirements,
CM_i = Num. of correctly matching templates in i-th iteration,
UR_i = Num. of user request in i-th iteration.

This experiment has been repeated for 10 different user requests over 100 times and recorded as the percentage of user requests that get Ideal Templates as shown in Figure 12 (a). The result confirms the intuitive phenomenon that as the information given to the recommender system decreases, there is lesser likelihood that the user receives the correct recommendation (about 75% drop in the accuracy for 10 templates). Another significant indicator to denote the disparity between the user requests and the ideal template selection is the Euclidean distance between them as shown in Figure 12 (b). The simulation conditions are described earlier and they display similar behavior with the distance metric showing about 2.25 times increase just for 10 templates, and results would be consistent (with amplification factors) for greater number of templates in a given catalog. The simulation space is scaled further to understand the impact on the percentage of the satisfied user requests as the catalog matures, thereby increasing in the number of saved templates. This is represented by the three different catalog sizes in Figure 12 (a) and (b) with template sizes as 10, 50 and 100. The observed results show that having multiple options available in the catalog does not necessarily improve the recommendation accuracy, as it falls with the increase in the number of templates in the catalog. In fact, even the best results have a substantial drop from 74.1% to 54% (with 2 missing parameters) in Figure 12 (a). This can be explained by the fact that the recommender has to work against the increased (catalog) space complexity.

7. Conclusion

In this paper, we presented that there are substantial barriers and a clear lack of scientific approaches and middleware solutions to help users of data-intensive applications to effectively deploy heterogeneous distributed cloud resources. We designed a user-friendly interface (KIS) to overcome limitations of reusability of previously successful configurations of resource provisioning for similar applications, thus demonstrating proper abstractions. Particularly, we described the implementation of a custom template catalog (i.e., our recommendation scheme contribution) that recommends configuration solutions for requirements of different applications, which in turn could lead to effectively utilize time and effort in provisioning heterogeneous resources for novice and expert users. The presented custom template middleware (i.e., our implementation for a real-world application, code openly available in GitHub at [40]) was evaluated through two experiment scenarios that involved simulation of novice beginner and expert interactions with the KIS. Results of the evaluation show that our scheme can obtain up to 71% accuracy for novice users, and up to 92% accuracy for expert users, thus a net improvement of 21% accuracy, compared to an existing recommendation scheme [26].

Our middleware also enables a provisioned heterogeneous cloud resource to be adapted and refined automatically after its provisioning in an online manner. We have designed an online iterative recommendation scheme with varying granularity of template modification (i.e., fine-grained and coarse-grained) to cover the possible performance disparity. We conducted a trade-off analysis between cost and performance for private to public cloud transformation for an advanced manufacturing science gateway to show that private clouds can provide cost-effective faster service solutions based on a selected custom template configuration. We also verified with simulation results that the maturity evaluation for novice and expert user with improvements in the accuracy of about 75% for 10 templates, and results would be consistent (with amplification factors) for greater number of templates in a given catalog.

Future work could involve enabling the questionnaire (content and flow) to be very close to the dynamic user-expert interaction. Thus, its parameters can be changed to multiple parameters mapping to particular CSP infrastructure features. The middleware could also enable rapid monitoring and sharing of performance data over larger group of templates to facilitate the needs of a user community. Lastly, our work could be extended to a number of other real-world applications to benefit data-intensive user communities in various science and engineering disciplines such as bioinformatics and neuroscience.

Acknowledgements

This work was supported by the National Science Foundation under awards: ACI-1246001, ACI-1245795, ACI-1440582 and CNS-1429294, and Cisco Systems. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or Cisco Systems.

References

[1] A. Hanemann, J. Boote, E. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, D. Swany, S. Troche, J. Zurawski, "perfSONAR: A Service Oriented Architecture for Multi-Domain Network Monitoring", *Proc. of Intl. Conference on Service-Oriented Computing (ICSOC)*, 2005.

[2] R. Morgan, S. Cantor, S. Carmody, W. Hoehn, K. Klingenstein, "Federated Security: The Shibboleth Approach", *EDUCAUSE Quarterly*, 27(4):12-17, 2004.

[3] R. Bazan Antequera, P. Calyam, S. Debroy, L. Cui, S. Seetharam, M. Dickinson, T. Joshi, D. Xu, T. Beyene, "ADON: Application-Driven Overlay Network-as-a-Service for Data-Intensive Science", *IEEE Transactions on Cloud Computing*, 2017.

[4] D. Jagli, S. Yeddu, "CloudSDLC: Cloud Software Development Life Cycle", *Intl. Journal of Computer Applications*, Vol. 168, No. 8, 2017.

[5] R. Schmidt, S. Grarup, "vApp: A Standards-based Container for Cloud Providers", *ACM SIGOPS Operating Systems Review*, 44(4):115-123, 2010.

[6] M. Berman, J. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, I. Seskar, "GENI: A federated testbed for innovative network experiments", *Elsevier Computer Networks*, 61(1):5-23, 2014.

[7] "Amazon Web Services: on-demand public computing platform"; https://aws.amazon.com
[8] R. Bazan, P. Calyam, A. Ankathatti, S. Malhotra, "Recommending Resources to Cloud Applications based on Custom Templates Composition", *Proc. in ACM Computing Frontiers Conference*, 2017.
[9] Terraform - Write, Plan, and Create Infrastructure as Code; https://www.terraform.io/
[10] A. Botea, M. Muller, J. Schaeffer, "Using Component Abstraction for Automatic Generation of Macro-Actions", *Proc. of Intl. Conference on Automated Planning and Scheduling (ICAPS)*, pp. 181-190, 2004.
[11] G. Wei, X. Zhong-Wei, X. Ren-Zuo, "Metrics of Graph Abstraction for Component-Based Software Architecture", *Proc. of the WRI World Congress on Computer Science and Information Engineering*, 2009.
[12] H. Zheng, H. Yao, T. Yoneda , "Modular Model Checking of Large Asynchronous Designs with Efficient Abstraction Refinement", *IEEE Transactions on Computers*, 59(4):561-573, 2010.
[13] Y. Wang, "A hierarchical abstraction model for software engineering", *Proc. of the International workshop on The role of abstraction in software engineering*, pp. 43-48, 2008.
[14] H. Qian, H. Zu, C. Cao, Q. Wang, "CSS: Facilitate the cloud service selection in IaaS platforms", *Proc. of Intl. Conference on Collaboration Technologies and Systems (CTS)*, 2013.
[15] T. Zain, M. Aslam, M. R. Imran, "Cloud Service Recommendation System Using Clustering", *Proc. of Intl. Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, 2014.
[16] M. Zhang, R. Ranjan, A. Haller, D. Georgakopoulos, and P. Strazdins, "Investigating decision support techniques for automating Cloud service selection", *Proc. of IEEE Intl. Conference on Cloud Computing Technology and Science (CloudCom)*, 2012.
[17] B. Zilci, M. Slawik, A. Kupper, "Cloud Service Matchmaking using Constraint Programming", *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2015.
[18] L. Liu, X. Yao, L. Qin, "Ontology-based Service Matching in Cloud Computing", *Proc. of IEEE Intl. Conference on Fuzzy Systems*, 2014.
[19] S. Sundareswaran, A. Squicciarini, D. Lin, "A Brokerage-Based Approach for Cloud Service Selection", *Proc. of IEEE Intl. Conference on Cloud Computing (CLOUD)*, 2012.
[20] M. Singhal, J. Ramanathan, P. Calyam, M. Skubic, "In-the-know: Recommendation Framework for City-supported Hybrid Cloud Services", *Proc. of IEEE/ACM Intl. Conference on Utility and Cloud Computing (UCC)*, 2014.
[21] Z. Rehman, F. Hussain, O. Hussain, "Towards Multi-Criteria Cloud Service Selection", *Proc. of Intl. Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2011.
[22] Z. Gui, C. Yang, J. Xia, Q. Huang, K. Liu, Z. Li, M. Yu, M. Sun, N. Zhou, B. Jin, "A Service Brokering and Recommendation Mechanism for Better Selecting Cloud Services", *PLoS ONE (Public Library of Science)*, 9(8):e105297, 2014.
[23] AWS CloudFormation - Infrastructure as Code & AWS Resource Provisioning; https://aws.amazon.com/cloudformation/details/#designer
[24] "Cisco UCS Director - automates, orchestrates, and manages Cisco and third-party hardware; http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-director
[25] R.Qasha, J. Cala, P. Watson,"Towards Automated Workflow Deployment in the Cloud using TOSCA" *Proc. of IEEE Intl. Conference on Cloud Computing (CLOUD)*, 2015.
[26] S. Soltani, P. Martin, K. Elgazzar, "QuARAMRecommender: Case-Based Reasoning for IaaS Service Selection", *Proc. of Intl. Conference on Cloud and Autonomic Computing (ICCAC)*, 2014.
[27] C. Horuk, G. Douglas, A. Gupta, C. Krintz, et. al., "Automatic and portable cloud deployment for scientific simulations", *Proc. of IEEE HPCS*, 2014.
[28] J. Kirschnick, J. Alcaraz, L. Wilcock, N Edwards, "Toward an architecture for the automated provisioning of cloud services", *IEEE Communications Magazine*, 2010.
[29] T. Nielsen, C. Iversen, P. Bonnet, "Private Cloud Configuration with MetaConfig", *Proc. of IEEE International Conference on Cloud Computing (CLOUD)*, 2011.
[30] Puppet - Utility to manage IT infrastructure as code across all environments; https://puppetlabs.com
[31] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment", *Linux Journal*, 239(2), 2014.
[32] C. Boettiger, "An introduction to docker for reproducible research", *ACM SIGOPS Operating Systems Review - Special Issue on Repeatability and Sharing of Experimental Artifacts archive*, 49(1):71-79, 2015.
[33] M. Thanh, N. Quang-Hung, M. Nguyen, N. Thoai, "Using Docker in High Performance Computing Applications", *Proc. of IEEE Intl. Conference on Communications and Electronics (ICCE)*, 2016.
[34] R. Qasha, J Cala, P. Watson, "A Framework for Scientific Workflow Reproducibility in the Cloud", *Proc. of 12th IEEE Intl. Conference on e-Science*, 2016.
[35] I. Giannakopoulos, N. Papailiou, C. Mantas, I. Konstantinou, D. Tsoumakos†, N. Koziris, "CELAR: Automated Application Elasticity Platform" *Proc. of IEEE Intl. Conference on Big Data (Big Data)*, 2014.
[36] Apache Cloud - a standard Python library that interfaces with multiple cloud providers; http://libcloud.apache.org
[37] Apache jCloud - an open source multi-cloud toolkit; http://jclouds.incubator.apache.org
[38] Delta-Cloud - comprises of an API server and drivers necessary for connecting to cloud providers; http://deltacloud.apache.org
[39] Cloudify - Cloud & NFV Orchestration Based on TOSCA; http://cloudify.co
[40] Custom Template Middleware - Openly accessible Github repository; https://github.com/acarjungowda/CustomTemplate_Recommender
[41] Y. Liu, S. Khan, J. Wang, M. Rynge, Y. Zhang, S. Zeng, S. Chen, J. Maldonado, B. Valliyodan, P. Calyam, N. Merchant, H. Nguyen, D. Xu, T. Joshi, "PGen: Large-Scale Pegasus Workflow for Genomic Variation Analysis in SoyKB", *BMC Bioinformatics*, 2016.
[42] X. Amatriain, A. Jaimes, N. Oliver, J. M. Pujol, "Data Mining Methods for Recommender Systems", *Springer, Recommender Systems Handbook*, 2010.
[43] A. Akula, P. Calyam, R. Bazan, R. Leto, "Advanced Manufacturing Collaboration in a Cloud-based App Marketplace", *Proc. of ACM Computing Frontiers Conference*, 2017.

Ronny Bazan Antequera received his MS degree in Computer Science from University of Missouri, Columbia in 2014. He received his BS degree in Computer Science from San Andres University, Bolivia. He is currently pursuing his Ph.D. degree in the Department of Electrical Engineering and Computer Science at University of Missouri-Columbia. His current research interests include Hybrid Cloud Computing, Network Security and Software-defined Networking.

Prasad Calyam received his MS and PhD degrees from the Department of Electrical and Computer Engineering at The Ohio State University in 2002 and 2007, respectively. He is currently an Associate Professor in the Department of Electrical Engineering and Computer Science at University of Missouri-Columbia. Previously, he was a Research Director at the Ohio Supercomputer Center. His current research interests include Distributed and Cloud Computing, Computer Networking, and Cyber Security. He is a Senior Member of IEEE.

Arjun Ankathatti Chandrashekara received his BS degree in Computer Science from Sri Siddharatha University, Bengaluru, India. He is currently pursuing his MS degree in the Department of Electrical Engineering and Computer Science at University of Missouri-Columbia. His current research interests include Cloud Computing, Recommenders, Expert Systems and Cognitive UI development.

Reshmi Mitra is a Post-doctoral Fellow in the Department of Electrical Engineering and Computer Science at the University of Missouri-Columbia. She received her MS and PhD degrees in Electrical and Computer Engineering from University of North Carolina at Charlotte in 2007 and 2015, respectively. Previously she has worked in National Institute of Technology India, Advanced Micro Devices Austin and Samsung Austin R&D Center. Her research interests include Interactive and Cognitive Cloud Computing and Performance Modeling.