Recommending Resources to Cloud Applications based on Custom Templates Composition

Ronny Bazan Antequera  
University of Missouri-Columbia, USA  
rcb553@mail.missouri.edu

Arjun Ankathatti Chandrashekarar  
University of Missouri-Columbia, USA  
aacwb@mail.missouri.edu

Prasad Calyam  
University of Missouri-Columbia, USA  
calyamp@missouri.edu

Shivoam Malhotra  
Indian Institute of Technology (Varanasi), India  
shivoam.malhotra.cse11@iitbhu.ac.in

ABSTRACT

Emerging interdisciplinary data-intensive applications in science and engineering fields (e.g., bioinformatics, cybermanufacturing) demand the use of high-performance computing resources. However, data-intensive applications’ local resources usually present limited capacity and availability due to sizable upfront costs. The applications requirements warrant intelligent resource ‘abstractions’ coupled with ‘reusable’ approaches to save time and effort in deploying cyberinfrastructure (CI). In this paper, we present a novel ‘custom templates’ management middleware to overcome this scarcity of resources by use of advanced CI management technologies/protocols to on-demand deploy data-intensive applications across distributed/federated cloud resources. Our middleware comprises of a novel resource recommendation scheme that abstracts user requirements of data-intensive applications and matches them with federated cloud resources using custom templates in a catalog. We evaluate the accuracy of our recommendation scheme in two experiment scenarios. The experiments involve simulating a series of user interactions with diverse applications requirements, also feature a real-world data-intensive application case study. Our experiment results show that our scheme improves the resource recommendation accuracy by up to 21%, compared to the existing schemes.

KEYWORDS

Federated Cloud Resources, Component Abstraction Model, Custom Templates, Recommendation Scheme

ACM Reference format:
DOI: http://dx.doi.org/10.1145/3075564.3075582

1 INTRODUCTION

Data-intensive science applications in fields such as bioinformatics, climate modeling, and particle physics are becoming increasingly multi-disciplinary. These 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 [35], 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 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 [4] (vApps) from VMware; Global Environment for Network Innovations (GENI) RSpecs [5] using XML files; and Amazon Machine Images [6] 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 approaches such as, VMware ThinApp, Citrix XenApp and Microsoft App-V that are limited in terms of platform and operating system independence. Thus, a suitable abstraction, virtualization, and orchestration approach is needed to fully comprehend the reusable heterogeneous distributed resource deployments for data-intensive applications.

In this paper, we address the above limitations of abstraction efforts and deployment approaches for heterogeneous distributed computing infrastructures. More specifically, we present a novel middleware, shown in Fig. 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 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.

We evaluate our middleware recommendation scheme with simulated interactions and a real-world data-intensive case study. We simulate a series of user interactions for diverse application 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 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 needs.

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

- Knowledge Interface System (KIS): The KIS efficiently utilizes different data structures to abstract requirements for data-intensive application requirements, and normalizes the data for future comparisons. It also significantly reduces the complexity of collecting requirements from novice and expert users, through intelligent interactions that guide them to effectively express their needs.
- Recommendation Scheme: This scheme takes into account the heterogeneity of distributed cloud resources, and recommends candidate solutions with high accuracy (up to 92% in best case scenarios). It does so by recommending custom templates that can be customized and stored in a catalog for future reuse.
- Resource Deployment Engine: This engine automates the process for cloud resources deployment in a federated-distributed cloud environment. The infrastructure is deployed by using any available cloud service provider (CSP) API, and software requirements of novice and expert users are met by using containerization.

The remainder 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

The related work can be broadly divided under into the following categories: resource abstraction models, knowledge interface & recommendation schemes, and middleware for automated distributed resource provisioning.

2.1 Component abstraction model

It is important to identify and abstract application requirements, as well as an available pool of resources to effectively provision infrastructure configurations. Hence, applying relevant methods for requirements abstraction is imperative. Our literature review found works related to this area such as [7], where authors use component abstraction for reducing the complexity in the planning and management processes. They break down a given problem by using heuristic rules in order to plan at the component level. In [8], 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 [9], authors apply abstractions to check large asynchronous designs based on component abstraction verification in isolation. This process eliminates the need for finding an accurate verification. Similarly, authors in [10] 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 abstraction and cloud infrastructure abstraction of distributed heterogeneous resources, an area which we found is still quite under-explored. We adapted the idea of component level abstraction to abstract distributed heterogeneous resources by enabling different domains such as networking component, storage component, computation component etc. as shown in Fig. 1. Thus, the decisions are made within
and across the components, which makes resource planning and composition relatively easy and more efficient.

2.2 Cloud service matching

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 find the best available cloud resources from multiple cloud service providers. The definition of the best resource is based on the user’s importance to the functional, non-functional and cost of the resources. Some research efforts consider the problem as a multi-criteria decision making (MCDM). For example, authors in [11] focus on recommendations for Software-as-a-Service (SaaS) using an Analytical Hierarchy Process to rank services; each service is ranked based on the weights collected by experts for each metric and the best service is recommended. A similar approach is presented in [12] through the implementation of a vector of cost, performance, and mobility generated for available resources, and a vector for user requirements in order to find resource similarity. In comparison, our work considers a recommender that accounts for several variables that should be considered such as metrics, QoS and feature importance that may vary from user to user.

Works that consider different variables for service matching and recommendation include [13] and [14]. In [14], a web ontology cloud model is constructed to abstract the heterogeneity of resource specification among multiple cloud service providers. The obtained solution is based on a genetic algorithm to match both qualitative and quantitative criteria. In [15], authors discuss the problem of selecting the closest available service to user data in order to decrease the transfer time and thereby, increase the response time. Work in [16] presents a cloud broker that discovers services and calculates the indexes for each service by considering user requirements. It then applies a K-NN algorithm to obtain services that match the user requirement based on the calculated indexes. Authors in [17] developed a comprehensive user interface for cloud service selection based on multiple criteria and considered user preferences (i.e., cost, QoS attributes). However, this approach lacks a deployment engine and functions as the equivalent of an e-commerce website for a CSP. Another approach is presented in [18], where authors discuss a system that uses a knowledge base to store available resources and previously recommended solutions. The knowledge base is searched for every new request to identify a similar solution that could be adapted to suit the new requirement. A knowledge base approach where user requirements are stored as a template in a catalog is also presented in [19]. In comparison, our work considers collection of novice and expert user requirements through intelligent KIS interactions and builds on the ideas to use a catalog and a corresponding knowledge base.

2.3 Automated resource provisioning frameworks

Studies on automation and provisioning of cloud resources have been conducted and presented in recent times. Authors in [23] 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. In [24], authors describe an architecture composed of four layers (Cloud infrastructure, Abstraction, Orchestration, and Design) that enables automated deployment of cloud services. It considers templates that encapsulate resource provisioning requirements. However, they do not consider novice and expert user interactions for requirements collection, and do not employ a recommender scheme to suggest a suitable solution, as we do in this paper.

In [25], the authors present MetaConfig system, a tool similar to Puppet [26] 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. Some studies on middleware to provision elastic resources for cloud application have been developed such as Celar [28] that allows users to select and deploy certain compute resources in different cloud platforms through API calls using abstraction libraries (such as Apache Libcloud [30], jclouds [31], delta-cloud [32]). Cisco UCS Director [29], a Unified Infrastructure Management middleware, also provisions compute, network and storage resources automatically in local and remote locations using a catalog equivalent scheme viz., ‘workflows’, without a recommender feature.

Our approach improves the abstraction and automated deployment process by applying intelligent catalog management and a recommender scheme that allows effective and easy to re-use existing solutions to save time, avoid guesswork and enable automated resource deployment over heterogeneous resources. Our work also considers container technology [33, 34] to foster reproducible experiments for data-intensive applications. Our middleware can be integrated into emerging heterogeneous distributed computing infrastructures and can be extended to complement other infrastructures that choose to adopt related standards such as OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [27].

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 Fig. 1 that involves three stages: Collection, Composition and Consumption. Fig. 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.

- **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 stage collects and abstracts user requirements through the KIS, which is a web-based solution that aims to mimic scenarios to collect requirements for data-intensive application infrastructure. Traditionally, the requirements collection is done through one-to-one
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: KIS is based on a 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.

The main function of the KIS is to collect user requirements for their applications and to help novice users to choose compatible resources. The first step collects applications’ high-level requirements and preferences to understand resource priority needs. This collected information helps the KIS to pre-populate relevant fields with default values by predicting the needs. However, users (novice and expert) can always change the pre-populated values to personalize their solution. The pre-population process uses a rule-based model (stored in a database) to select the most adequate answer (based on previous interaction with the KIS) which guarantees minimum functional resource compatibility for a given set of application requirements. The KIS additionally allows users to reuse previously deployed templates (select or upload template) in order to customize them and skip the questionnaire process.

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 (step 2 of Fig. 2). 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 is shown in Fig. 4, where $F_l$ represents required resource feature (i.e., Bandwidth, Storage Size, RAM size) and $P_l$ represents the corresponding required precondition (i.e., 20 Mbps bandwidth, 20 GB HDD storage, 8 GB RAM). A single or multiple such $F_l - P_l$ pairs (i ∈ I, where I is the set of required resources, and I = |I| is the number of elements in 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.

**Figure 4: Application Requirement Identifier structure**

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 (step 3 of Fig. 2) are categorized into domains, i.e., VPN (virtual private network) and network bandwidth resources belong to the overall Network Connectivity domain and No. of CPUs cores, RAM size belong to the overall Computation domain.

We define $D$ as the set of all cloud resource domains where $D = \{D_1, D_2, ..., D_N\}$, and $N$ is the number of domain categories the cloud resources are divided into $N = |D|$. Each such domain consists of a different number of cloud resources as follows:

$$
D_1 = \{R_1^1, \ldots, R_A^1\} \text{ where } A \text{ is the no. of resources in } D_1
$$

$$
D_2 = \{R_2^1, \ldots, R_B^1\} \text{ where } B \text{ is the no. of resources in } D_2
$$

$$
:\text{ }
$$

$$
D_N = \{R_1^N, \ldots, R_N^N\} \text{ where } N \text{ is the no. of resources in } D_N
$$

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 and their corresponding performance bounds of the respective resources and the associated cost.

The collected data is then compared with the RSpace data structure is shown below. The RSpace data structure is represented as:

$${R_1}^A = \{S_{11}^1, C_{11}^1, \ldots, S_{1A}^1, C_{1A}^1\}$$

$${R_2}^A = \{S_{11}^2, C_{11}^2, \ldots, S_{1B}^2, C_{1B}^2\}$$

$$\vdots$$

$${R_A}^A = \{S_{11}^A, C_{11}^A, \ldots, S_{1N}^A, C_{1N}^A\}$$

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 $A \text{ to } N$ are the number of constraints in $R_1^A$ to $R_A^1$ respectively. Such a formalization maintains the uniqueness of each domain, resource, constraint and its corresponding resource cost.

Fig. 5 shows RSpace representation of resources and their corresponding constraints categorized in different domains.

**Figure 5: Resource Space abstraction**

3.1.3 *Macro Operator*. The output data structure of the Collections stage is called as the Macro Operator (MacOps), 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 $D$, resource couples in RSpace and are represented as:

$$
\begin{align*}
    f : (F_l, P_l) & \mapsto (D_k, R_j^k) \\
    \text{where } l \in I, k \in D, \text{ and } j \in \{\hat{A} | \hat{B} | \ldots | \hat{R}\}. \text{ Here } \hat{A} \text{ denotes the set with } \hat{A} \text{ resources. Equation (1) formulates the process of the ARI features translating into domains and the corresponding ARI preconditions along with RSpace resource specifications and constraints being mapped into one or multiple resources in that domain, forming one or multiple (Domain, Resource) pairs. One limiting criteria for such a mapping is that the total number of l’s should be equal to the total number of k’s, i.e., each feature in an ARI maps to one and only one domain in RSpace. However, no such restrictions apply for a number of resources within the domain. The corresponding ARI preconditions are added to the domain-resource pairs forming MacOps as (Domain, Resource, Precondition, Cost) 4-tuples.}
\end{align*}
$$

A MacOp representation is shown in Fig. 6 that corresponds to an ARI (Feature, Precondition) couples shown in Fig. 4. For this particular example, we see that ARI ($F_l, P_l$) has been mapped to multiple MacOps, viz. ($D_1, R_1^1$), and ($D_1, R_2^2$) denoting either can satisfy the ARI.

As we mentioned, a single (Feature, Precondition) tuple in ARI can lead to multiple Resources in MacOps belonging to the same domain. One such example of a MacOps generation is shown in Table 1, where ‘Network Connectivity’ domain have multiple resource options that can solve the network connectivity requirements of the application.

**Figure 6: Macro Operator**
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 generated using all possible combinations of MacOps (a set of candidate cloud resources that could satisfy data intensive application requirements, equation: 1) and then stored in the catalog. 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). Fig. 7 shows an excerpt of the custom template JSON file for an example application that contains a sample description of cloud resource configuration and application metadata.

3.2.2 Catalog. As the new templates are created they are stored in a catalog for future use (step 4 in custom template life cycle). 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 time the catalog will have critical mass with new custom templates upon deployment of new resources.

3.2.3 Policy Database. We consider Policy database which contains a pre-defined set of rules with inter-dependencies among them. They are used to provide initial resource recommendation for the KIS. 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.

Once the user chooses a particular template (Step 8 in Fig. 2), 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 for software deployment i.e., through the ‘docker hub’. Once the infrastructure is deployed, the monitoring manager (Step 9 in Fig. 2) 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 data 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

In this section, we present details of our template recommendation algorithm that finds the most closely matching templates for a given user’s requirement. 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 ARI, 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 MacOp data structure contains different candidate resources for different domains. This is used to compose templates from all possible combinations and to store unique templates in the catalog. A filter process for string comparison is applied to the catalog and the reqVector to find candidates templates that contain the required cloud resources. Next, the distance between candidates and reqVector is calculated and stored in a list (tList). A second filtering process is
Algorithm 1 Template recommendation algorithm

1: procedure DECISION(MacOp[1...m], catalog[1...n], pDimension, reqVector)
2: /*number of neighbors*/
3: k = 3
4: /*generate possible candidates combination of MacOp*/
5: m = MacOpComb(MacOp)
6: /*update catalog with generated candidates*/
7: catalog = catalog + m
8: /*filter catalog based on string values*/
9: f = filter(catalog, reqVector)
10: /*calculate templates distance to reqVector*/
11: for all t in f do
12: tList = distance(t, reqVector, pDimension)
13: end for
14: /*filter k number of candidate templates*/
15: candidateList = nearestNeighbors(k, tList, reqVector)
16: /*select closest template*/
17: green = closestT(candidateList, reqVector)
18: /*select max preferred dimension template*/
19: gold = maxPDimension(candidateList, pDimension)
20: /*select cheapest Template*/
21: red = cheapestT(candidateList)
22: return red, gold, or green
23: end procedure

applied to the catalog i in order to find the 3 closest templates using the K-NN algorithm.

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 2 for \( R_i \in \) RSpace.

\[
D_i = w \sum_{j=1}^{n} V_j \cdot \text{abs}(R_{i,j} - V_j) + w \cdot \sum_{j=1}^{n} V_j \cdot \text{max}(0, R_{i,j} - V_j).
\]  

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

We consider the closest solution i.e., the one with the least distance to the reqVector as the ‘green’ template (function closestT() in the Algorithm), and the one with high value for the pDimension as ‘gold’ template (function maxPDimension() in the Algorithm) and the remaining candidate as ‘red’ template (function cheapestT() in the Algorithm). We also consider range of resource requirements to determine upper and lower boundaries as the KIS allows ranges for certain resources requirements.

5 IMPLEMENTATION AND RESOURCE DEPLOYMENT

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 is done by using Docker functionality 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, data and GENI configuration openly available at [36].

Fig 8 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 Manager’ verifies that the resources are deployed successfully. All the inter-process communication between different layers is implemented via RESTful web services.

![Diagram](image-url)
manager sends back the rest of the form with pre-populated values which are minimum requirements for the application. The complete user requirements are sent to solution generator which generates the ARI and sends it to the MacOp generator. The MacOp generator compares the ARI with the RSpace and gives the possible resources in each component that can satisfy the user requirement. The solution generator then generates all possible combinations of the solution and creates the MacOps and inserts them into the catalog. Here, only the unique templates are inserted into the catalog, to avoid duplicates. Next, the solution generator contacts the recommendation scheme to output 3 templates (‘green’, ‘red’ and ‘gold’) that are the 3 closest solutions to the user requirement as per Algorithm 1. The templates are shown to the user and upon user selection, the selected template (among ‘green’, ‘red’ and ‘gold’) is sent to configuration manager for deployment. Lastly, access information of the configured resources and credentials are available to the user.

We implemented our middleware to provision cloud infrastructure for a real-world advanced manufacturing data-intensive application. Advanced manufacturing design requires synchronous collaborative work among multi-site engineering experts and a robust local infrastructure that can run a highly scalable workflow. Small and Medium size manufacturer businesses are increasingly leveraging modeling and simulation processes as they cannot afford to invest in local infrastructures. We observe that our middleware implementation reduces the product design cycles and provides the businesses with expertise to manage the sizable upfront costs for high-performance computing resources and software licenses.

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 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.

6.1 Evaluation of Recommendation System

For our evaluation experiments, we used the publicly available distributed infrastructure of GENI. We setup a testbed with distributed heterogeneous resources that can be discovered and configured by our recommendation scheme using the geni-lib, a GENI API capability. To mimic the different data centers, we considered different aggregator managers (configurable computational resources) that are distributed across many cities.

To evaluate our middleware utility, we use a Resource Provisioning 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 system 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 [18] that also has a recommendation approach as mentioned in Section 2.

Fig 9(b) shows results obtained with our scheme which clearly presents improved results in comparison with Fig 9(a). Results in 9(b) 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 Data-intensive Application Case Study

In this subsection, we present the effectiveness of our middleware implementation in the form of a case study for the advanced manufacturing of data-intensive application.

Case study: A small-business advanced manufacturing company integrated our custom template middleware solution and provided their cloud resource requirement 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 GB-40 GB local 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

Some of the cloud resource requirements (RAM, Storage Size and Bandwidth) present 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 $V_{max} = (2, 8,$
Figure 9: Accuracy for recommended results: (a) This figure presents results based on Figs. 5 and 6 of work done in [18], 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%.

Figure 10: 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. (a) pDimension is not specified hence some templates are out of the boundaries; (b) 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.

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’s and presents the available resources to the user for 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. 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 Fig. 10, 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 Fig. 10(a) show that the cloud resources are out of the boundaries. However, results in Fig. 10(b) 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 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.

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 [36]) 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 [18].

Future work could involve enabling dynamic resource adaptation over federated cloud resources based on resource utilization monitoring. Also, middleware extensions can be developed to allow a provisioned distributed cloud resource to be adapted and refined automatically after its provisioning in an online manner. These extensions could solve cases e.g., when a user is not sure about the required resources or the KIS inputs are scarce. 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.

Acknowledgments

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


[6] “Amazon Web Services: offers a suite of cloud computing services that make up an on-demand computing platform”, http://aws.amazon.com


[26] Puppet: Manage IT infrastructure as code across all environments”, https://puppetlabs.com


[29] “Cisco UCS Director: Cisco UCS Director automates, orchestrates, and manages Cisco and third-party hardware. It allows a user to manage network components and services and allows data center staff to focus on services.” http://www.cisco.com/en/us/products/ucsserver/multicomputing/ucs-director/index.html

[30] “Apache Cloud: it is a standard Python library that abstracts away different multiple cloud providers i.e. it provides one interface to interact with all CSPs.” http://cloud센터.apache.org

[31] “Apache jCloud: it is an open source multi-cloud toolkit for the Java platform that gives users the freedom to create applications that are portable across clouds while giving you full control to use cloud-specific features.” http://clouds.incubator.apache.org

[32] “Delta-Cloud: Deltacloud provides the API server and drivers necessary for connecting to cloud providers. It enables management of resources in different clouds by the use of one of three supported APIs.” http://deltacloud.apache.org


