Reduced Complexity Of Service Matching Using Tree Concept For Multi-Tenants In Cloud Environment

Dhara D. Virani 1, Khushbu D. Virani 2, Madhuri Vaghasia 3

1 PG Student, Marwadi Education Foundation Group Of institutions, Rajkot, Gujarat.
2 PG Student, Marwadi Education Foundation Group Of institutions, Rajkot, Gujarat.
3 Asst. Prof., Marwadi Education Foundation Group Of institutions, Rajkot, Gujarat.

I. INTRODUCTION

The Greek myths tell of creatures plucked from the surface of the Earth and enshrined as constellations in the night sky. Something similar is happening today in the world of computing. Data and programs are being swept up from desktop PCs and corporate server rooms and installed in the compute cloud. In general, there is a shift in the geography of computation what is cloud computing exactly? “An emerging computer paradigm where data and services reside in massively scalable data centers in the cloud and can be accessed from any connected devices over the internet” Cloud computing is an emerging paradigm in the computer industry where the computing is moved to a cloud of computers. It has become one of the buzz words of the industry. The core concept of cloud computing is, quite simply, that the vast computing resources that we need will reside somewhere out there in the cloud of computers and we’ll connect to them and use them as and when needed.

II. THEORETICAL BACKGROUND

A multi-tenant application lets customers (tenants) share the same hardware resources, by offering them one shared application and database instance, while allowing them to configure the application to fit their needs as if it runs on a dedicated environment. The key aspects of multi-tenancy:
1. The ability of the application to share hardware resources.
2. The offering of a high degree of configurability of the software.
3. The architectural approach in which the tenants (or users) make use of a single application and database instance.
2.1. Introduction to Service Matching

System allows the client to state explicitly their functional and non-functional requirements; there are times when a client is unable to provide such information. Therefore propose a technique that helps create platform and infrastructural assignments for a new tenant. Since functionalities that match a client’s requirements can come from multiple existing tenants with different specifications, platform and infrastructural matching for a new tenant can be determined by combining values for features such as operating systems, software frameworks, memory requirement from across all matched tenants.

2.2. Approach to Taxonomy Based Service Matching

There are several open issues with cloud such as security, availability, scalability, interoperability, service level agreement, data migration, data governance, trusty pyramid, user centric privacy, transparency, political and legal issues, business service management etc. In the ETE (Enterprise Requirement-Tenant Supplier- End User Requirement) Architecture, different types of services considered at user and tenant provider side. After consideration match the user requirements with Tenant functionality.

2.3. Complexity Analysis of Existing Technique

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Technique</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Force Algorithm</td>
<td>String Matching</td>
<td>O(mn) where m=n=string length</td>
</tr>
<tr>
<td>Hungarian algorithm-Bipartite</td>
<td>Graph Matching</td>
<td>O(</td>
</tr>
<tr>
<td>Chunk Folding</td>
<td>Schema Matching</td>
<td>Change Vary over time</td>
</tr>
<tr>
<td>Tenant selector matching using</td>
<td>Taxonomy Matching</td>
<td>O(n³)</td>
</tr>
</tbody>
</table>

III. PROPOSED ALGORITHM

3.1. Introduction to System

The taxonomy has a tree-based structure. At the root of the tree are all cloud services. The first level is made up of the three main service categories. The next levels correspond to the common characteristics, followed by the service specific characteristics.

The taxonomy levels are
1. Service Category-S_c
2. License Type-L_t
3. Payment System-P_s
4. Service Level Agreement-S_a
5. Standardization Effort-E_s

In the tree based structure, all levels are particular defined for a specific type. All levels are increase the performance and it is useful for service matching. Additionally, the cloud can be expanded to include a grading of importance scheme. For matching cloud computing use degree of matching. The here proposed taxonomy is capable of classifying both current and future cloud computing services. The simple tree structure allows quick comparisons, by giving the user a set of choices at each level. This clear structure makes comparing cloud computing services more efficient than using table based comparisons.

3.2. Proposed Algorithm

Tree Level Service Matching(TLSM)

The proposed algorithm is as follows.
Step 1: Get End Users Service Requirement.
Step 2: Store Requirement according to User ID.
Step 3: Get Tenant List T which is used for Cloud.
// Find Available tenant which is free
Step 4: for i = 1 to n do
Step 5: Check if Availble Tenant T_i has an PAUSED State
Then
    Change in ACTIVE State.
Else
    TENANT has BUSY state.
end for
Step 6: Generate List of AVAILABLE Tenant ID
Step 7: Create TREE For Both Tenant Services for a specific Tenant Ti and
    Users Services for Particular User ID Ui.
Step 8: TREE Create with Specific Level   // Creating a Tree
    for TREE Level Ti = 1 to j do
    T[i][j] = {Sc, Lt, Ps, Sa, Es}  // Different levels
end for
Step 9: Now Matching the Tree Ti and Ui
Step 10: Initialization: (w1, w2, w3) ← weight of Tenant nodes
    // Tree node mapping
Step 11: if the root of trees A and B contain different symbol
    then
Step 12: return (0, A.nodes, B.nodes)  // Services are not Match
Step 13: end else
Step 14: Uj←No. of subtree for User service Level of A;
Step 15: Pj←No. of subtree for Provider service Level of B;
Step 16: Initialization: a[i, 0] ← 0 for i = 0, …, Uj;
Step 17: b[0, j] ← 0 for j = 0, …, Pj;
Step 18: for each Tenant Ti   // Degree of matching
Step 19: for each Tree level Lj
Step 20: Match Us and Ps
Step 21: if pair of node ([A, B]⇒ Exact)then w(A, B)⇒ w1
Step 22: if pair of node ([A, B]⇒ Plug in)then w(A, B)⇒ w2
Step 23: if pair of node ([A, B]⇒ Subsume)then w(A, B)⇒ w3
Step 24: elseif pair of node ([A, B]⇒ fail)then w(A, B)⇒ 0
    Break;
Step 25: end for
Step 26: end for
Step 27: for each Tenant Ti
Step 28: Ti = \sum_{j=1}^{k} Weight of node j
Step 29: Find Max-weight Tenant Ti in all tenants
Step 30: Assign a Tenant Ti to User.
Step 31: end for

3.3. Description of Algorithm

The TLSM algorithm which handles nested tree service matching list is given in Figure 5, where
A and B are trees to be matched. It follows the formulation of Service Tree Matching(TLSM) above,
Lines 1-2 in Figure 5 Get the Users and tenants requirement. Lines 4-6 find the available tenant
which are free. Lines 7-8 generate a tree according to specific level. Lines 11-12 compare the labels (tag
names) of the root nodes of the two trees A and B and return 0 if they are different. Lines 14-17 initialize
variables. Lines 18-26 compare the degree of matching and assign a weight to each nodesLines 27-29 sum
of weight and find maximum weightage tenant. There are specific level for a tree and it is defined in Lines 8
{Sc, Lt, Ps, Sa, Es} which are Sc= Service Category, Lt = License type, Ps = Payment System, Sa = Service
Level Agreement,Es = Standardization Effort.

When the degree of match between a client requirement and a tenant functionality is to be
calculated, their inputs and outputs are compared to identify the best match for each input and output
parameters of a client requirement. Degree of match can be one of exact match, instance of (plug-
in),subsumption or disjoint. Matches are ordered according to their degree of similarity,that is, exact match
> plug-in > subsume > disjoint. The best match between a client requirement and a tenant functionality is
determined by taking a minimum of their input and output matches.
Exact: Client’s Requirement = Tenant’s Functionality

Plug-in: A plug-in match is one where inputs and/or outputs of a client requirement form a supertype of the inputs and/or outputs of a tenant functionality.

Subsume: In case of a subsume match, a tenant functionality’s inputs and/or outputs form a supertype of a client requirement’s inputs and/or outputs.

Fail: Client’s Requirement ≠ Tenant’s Functionality

3.4. Complexity Analysis of Proposed Algorithm and Results

Complexity of the TLSM (Tree Level Service Matching) Algorithm calculated with the inclusion of several tasks such as searching and matching input and output tenant functionality. Here calculating relationship between existing tenants and user functionalities and iterating through all the tenants and determine number of matching and find a execution time which have a running time of $O(N^2)$. Thus. Complexity of the Algorithm in terms of its most time consuming task is $O(N^2)$.

Step 1: Let $N$ denote the Number of Tenant which is available in the cloud. So First Check for all tenant Which is in ACTIVE state. So $N$ times execute.

Step 2: Now Create a Tree with a specific level for all tenants which is above defined in TLSM Algorithm. So again it $N$ times perform.

Step 3: Initializing the weights $w_1, w_2, w_3$ which is assign to a particular node based on the Degree of Matching which is constant. Hence, $O(1)$ time execute.

Step 4: Define array of User Level Subtree and Service Provider Level Subtree individually so both $O(1)$ time execute.

Step 5: Now Matching a functionality of User Tree with all $N$ Tenants and which is also match with all specified tree level. So $N^2$ times execute.

Step 6: Find a Maximum weight of tenant from all available tenants. Hence, $N$ times perform.

Now Calculating the Time Complexity is Simplified.

$$T(N) = N+N+1+1+4+N^2+N$$
$$= N^2 + 3N +3$$
$$= O(N^2)$$

3.5 Comparative Analysis with Existing Techniques

Table 2 Comparative analysis with existing techniques

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Technique</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Force Algorithm</td>
<td>String Matching</td>
<td>$O(mn)$ where $m=n=string length$</td>
</tr>
<tr>
<td>Hungarian algorithm Bipartite graph matching (Dynamic)</td>
<td>Graph Matching</td>
<td>$O(</td>
</tr>
<tr>
<td>Chunk Folding</td>
<td>Schema Matching</td>
<td>Change Vary over time</td>
</tr>
<tr>
<td>Tenant selector matching using Degree of match in Cloud environment</td>
<td>Taxonomy Matching</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Tree Level Service Matching in Cloud environment</td>
<td>Taxonomy Matching</td>
<td>$O(N^2)$</td>
</tr>
</tbody>
</table>

3.6. Conclusion And Future Work

Platform as a service (PaaS) and Software as a service (SaaS) are find the crucial issue of dynamic, service-tenant in cloud-based systems, and it containms several key features specifically suited to cloud-based systems like integrated functional and non-functional requirement matching at SaaS, PaaS and IaaS levels;
Dynamic resource allocation using state information, and elimination of redundant tenant functionalities in order to prune the search space. And also cost and time are reduced to matching a service and dynamic resource allocation. Match making Algorithm choose an appropriate tenant according to users requirement however, it find based on the degree of match, states, constraints and behavior of tenant with tree design. Future work would involve using different tree based approach and testing in larger case studies and real-life cloud based systems from many different domains.

REFERENCES


