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.*

ABSTRACT:

Cloud computing is a highly Research area in the technical I.T field and economic world, and many of the software industry have entered the development of cloud services. It is the preeminent on-demand service system along with a Pay-as-you-go Policy. In Multi-tenant networking, with which multiple customers (tenant) networks are virtualized over a single collective physical infrastructure. Dynamic provisioning in the cloud requires an integrated solution across the technology stack (software, platform and infrastructure) combining functional, non-functional and resource allocation requirements. Research works in the area of web service matching. It reviews the available cloud computing services and identifies and clarifies their main characteristics. The Architectural features of multi-tenancy and classify them according to the requirements of end-users, enterprise that use the cloud as a platform, and tenant providers themselves. Service is matched with existing tenants and according to the requirement of end-users. Matching techniques such as string-based, chema based, semantic web service based, constraint-based, linguistic, graph-based and taxonomy-based. Clients spend extreme amounts of time and energy searching through a list of available services.

KEYWORDS: *Cloud Computing,Complexity, Multi-tenant, Service Matching, Taxonomy based Services*

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 1 Complexity Analysis of Existing Technique

Algorithm	Technique	Time Complexity
Brute Force Algorithm	String Matching	O(mn) where m=n=string length
Hungarian algorithm-Bipartite graph matching (Dynamic)	Graph Matching	$O(\|v\|^3)$ |V|= no. of vertices in graph
Chunk Folding	Schema Matching	Change Vary over time
Tenant selector matching using Degree of match	Taxonomy Matching	$O(n^4)$

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 $T_t$ and
                   Users Services for Particular User ID $U_t$.
Step 8:  TREE Create with Specific Level              // Creating a Tree
            **for** TREE Level $T_L = 1$ to j do
            $T[j] = \{S_c, L_t, P_s, S_a, E_s\}$           // Different levels
            **end for**
Step 9:  Now Matching the Tree $T_t$ and $U_t$
Step 10:Initialization:  $(w_1, w_2, w_3) \leftarrow$ weight of Tenant nodes
            // Tree node mapping
Step 11:**if** the roots of trees A and B contain different symbol
            **then**
Step 12:**return** (0, A.nodes, B.nodes)  // Services are not Match
Step 13:**else**
Step 14:     $U_s \leftarrow$No. of subtree  for User service  Level of A;
Step 15:     $P_s \leftarrow$No. of  subtree for Provider service  Level of B;
Step 16:Initialization:  $a[i, 0] \leftarrow 0$ for $i = 0, \ldots, U_s$;
Step 17:                         $b[0, j] \leftarrow 0$ for $j = 0, \ldots, P_s$;
Step 18:**for** each Tenant $T_i$              // Degree of matching
Step 19:**for** each Tree level $T_L$
Step 20:    Match Us and Ps
Step 21:          **if** pair of node ($[A_i , B_j]$= Exact)**then** $w(A_i , B_j) = w_1$
Step 22:          **if** pair of node ($[A_i , B_j]$= Plug in)**then** $w(A_i , B_j) = w_2$
Step 23:          **if** pair of node ($[A_i , B_j]$= Subsume)**then** $w(A_i , B_j) = w_3$
Step 24:          **elseif** pair of node ($[A_i , B_j]$= fail)**then** $w(A_i , B_j) = 0$
                     Break;
Step 25:**end for**
Step 26:**end for**
Step 27:**for** each Tenant $T_i$
Step 28:$T_i$ =       $\sum_{i=1}^{n}$(Weight of nod
Step 29:Find Max-weight Tenant $T_i$ in all tenants
Step 30:Assign a Tenant $T_i$ 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 $\{S_c, L_t, P_s, S_a, E_s\}$ which are S_c= Service Category, L_t = License type, P_s = Payment System, S_a = Service Level Agreement,E_s= 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 forma 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 excecution 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: Initiallizing the weights *w1,w2,w3* 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.

$$\begin{aligned} T(N) &= N+N+1+1+1+N^2+N \\ &= N^2 + 3N + 3 \\ &= O(N^2) \end{aligned}$$

3.5 Comparative Analysis with Existing Techniques

Table 2 Comparative analysis with existing techniques

Algorithm	Technique	Time Complexity
Brute Force Algorithm	String Matching	O(mn) where m=n=string length
Hungarian algorithm Bipartite graph matching (Dynamic)	Graph Matching	$O(\|v\|^3)$ |V|= no. of vertices in graph
Chunk Folding	Schema Matching	Change Vary over time
Tenant selector matching using Degree of match	Taxonomy Matching	$O(n^4)$
Tree Level Service Matching in Cloud environment	Taxonomy Matching	$O(N^2)$

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 containsn 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 inlarger case studies and real-life cloud based systems from many different domains.

REFERENCES

[1] C. N. Höfer and G. Karagiannis, "Cloud computing services: taxonomy and comparison", Journal of internet services and application, vol.2, Number 2, 2011, 81-94.

[2] Grobauer, B.walloschek, T.stocker, "Understanding cloud computing Vulnerabilities", Security and privacy IEEE, vol. 9, 2011, pp.50-57.

[3] L. Ramachandran, N. C. Narendra, K. Ponnalagu, "Dynamic provisioning in Multi-tenant Service clouds", Service Oriented Computing and Applications, Vol. 6, 2012.

[4] A. carus, H. Nusret bulus, A.mesut, "wordmatch: word based stringmatching over compressed texts", international scientific conference, Nov. 2007, vol. I, 357-360.

[5] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, J. Rittinger, "Multi-tenant Databases for Software as a service: schema mapping techniques", SIGMOD'08, June 9–12, 2008.

[6] D. Skoutas, A. Sheth, "Efficient Semantic Web Service Discovery in Centralized and P2P Environments", ISWC 2008, pp. 583–598.

[7] U. Bellur, R. Kulkarni, "Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching", IEEE International Conference onWeb Services, 2007, pp. 86 – 93.

[8] M. Paolucci, T. Kawamura, TR. Payne, KP. Sycara(2002), "Semantic matching of web services capabilities". In: International semanticweb conference, pp 333–347.

[9] R. Mietzner, A. Metzger, F. Leymann, K. Pohl (2009), "Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications". In: PESOS '09: proceedings of the 2009 ICSE workshop on Principles of engineering service oriented systems. IEEE Computer Society, Washington, DC, USA, pp 18–25.

[10] M. Hachicha; J. Darmont, "A Survey of XML Tree Patterns," Knowledge and Data Engineering, IEEE Transactions on , vol.no.99, pp.1-20, oct-2011.

[11] U. Bellur, H. Vadodaria (2008) On extending, "semantic matchmaking to include preconditions and effects". In: ICWS, pp 120–128.

[12] R.Calheiros, R.Ranjan, A. Beloglazov, César A. F. De Rose, and Rajkumar Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms", Software: Practice and Experience, Vol. 41, issue 1, 2011, pp 23-50.