A Hypergraph Partition Based Approach to Dynamic Deployment for Service-Oriented Multi-tenant SaaS Applications

Ying Pan, Lei Wu, Shijun Liu, Xiangxu Meng

A Hypergraph Partition Based Approach to Dynamic Deployment for Service-oriented Multi-tenant SaaS Applications

Ying Pan , Lei Wu[+] , Shijun Liu, Xiangxu Meng

School of Computer Science and Technology, Shandong University
250101, Jinan, P.R.China
panying0501@ mail.sdu.edu.cn, {i_lily, lsj, mxx }@ sdu.edu.cn

Abstract. In a service-oriented multi-tenant SaaS application, all tenants share services and user requests of the service change dynamically. In order to provide high-quality web services, we must solve the problem of the load unbalance caused by dynamic user requests' change. This paper proposes an approach based on hypergraph partition to keep load balance for service-oriented multi-tenant SaaS application. A hypergraph-based service model is used to present hierarchical services and multi-tenant applications. This approach adjusts service distribution on the servers based on hypergraph partition to keep load balance. According to the experiments, this approach effectively solves the problem of load unbalance caused by the change of user requests.

Keywords: service deployment; hypergraph partition; multi-tenant; hierarchical services; SOA;

1 Introduction

Service-Oriented architecture (SOA) has become the standard for enterprise application development and integration. In SOA model, many kinds of computing resources (e.g, applications, services, and servers) are provided as services [1]. It has been considered as an effective solution for enterprise users because of its low cost and rapid reuse [2]. SaaS (Software as a Service) providers usually develop and acquire software applications and host them as services to serve specific requests of their clients.

In service component based SaaS applications, each multi-tenant application consists of a series of services while services are shared by multiple tenants. The multi-tenant applications improve profit margin for both service providers and enterprise users through reducing delivery costs and decreasing service subscription. However, it also introduces a noticeable problem that how to guarantee the quality of service (QoS) for multiple tenants who share applications. If the server that services deploy on is overloaded, the QoS for tenants will be sharply down. In the global

[+] To whom correspondence should be addressed.

consideration, the QoS for each tenant can be mapped to the loads of services that consist the multi-tenant application, that is to say, we should keep the server balance in order to meet the Qos for tenants. Additionally, the server load changes with user requests dynamically.

Therefore, in this paper we propose an approach based on hypergraph partition to solve the problem of dynamic service deployment under the premise of load balance. The approach is designed to ensure that SaaS providers can deploy and manage the service instances with the dynamic change of multiple tenants' service requests. A hypergraph-based service model is used to represent hierarchical services and these services are divided into several partitions. We propose algorithms that map each hypergraph partition to a server to deploy and balance each partition according to user requests change.

The structure of our paper is organized as follows. In section 2, we outline hypergraph partition-based service deployment. This is followed by section 3 that describes our algorithms for dynamic service deployment. We give some discussions about the overall performance of the approach according to experiments in section 4. Finally, conclusions and future work directions will be shown in the last section.

2 Hypergraph Partition Based Service Deployment for Multi-tenant SaaS Applications

2.1 Hypergraph-based Service Deployment Model

We consider various business requirements and the dependencies that exist in multi-tenant applications and services, thus services are divided into three levels in our research [8]: business-independent level, business-dependent level and composite business level. There are two kinds of dependences between the services, including functional dependency and business dependency. Such dependences widely exist in tenant applications and services, services in different and identical levels. But traditional DAG is hardly able to represent such structure and dynamic change of hierarchical services. Therefore, we introduce hypergraph partition theory into the service deployment. We establish directed hypergraph-based service deployment model to map servers.

Definition 1 (service): A service is a tuple S = (sId, subServices, sDes, sLevel), where: sId is the identifier of service; subServices is a finite set of services that the service depends on; subServices = { s_1, s_2, … , s_m }, where each s_i = (target, tRefType), target∈S, tRefTye is the type of dependence which including functional dependency and business dependency; sDes is the description of service; sLevel is the level of service, including business-independent level, business-dependent level and composite business level.

Definition 2 (tenant application): A tenant application is a tuple T = (tInfo, Services, tQos， tNonFun), where: tInfo is the basic information of tenant application, including the tenant information and so on; Services = {S_1, S_2,⋯, S_n} is a finite set of

services; tQos is the QoS for the tenant; tNonFun is non-functional properties of the tenant.

Definition 3: A service deployment directed hypergraph is denoted as SDDG = <V, E>, where:

1) V = {v_1, v_2, ⋯, v_n} is a finite set, where: each v_i ($i \in [1,n]$) corresponds to a service s_i, while v_i = {S, sR, sG, sQos, sGain, sType}.

- S is a service which is rented by multi-tenants.
- sR = {sr_1, sr_2, …, sr_k} is a finite set of resources that the service need.
- sG = {sg_1, sg_2, ⋯, sg_k} is a finite set of resource quantities which sg_i corresponds to sr_i ($i \in [1,k]$).
- sQos is the quality of service.
- sGain is the gain value of service. It is the important basis for vertex movement in hypergraph partition. In this paper, sGain represents comprehensive value of resource consumption computed by the resource consumption estimation model. The higher the value, the greater it impacts the partition. It indicates that the vertex has higher priority during the current partition adjustment.
- sType is the type of vertex, including BV and FV. BV is a base vertex, which is chosen to move to another partition. FV is a free vertex except to base vertexes.

2) E = {e_1, e_2, ⋯, e_m} is a finite set of hyperedges, m = |E| is the number of hyperedges, eq = { sPre, v_i},($q \in [1,m]$)

- sPre is a non-empty subset of vertices v_j, which $v_j \in V$. It indicates that the vertex v_i depends on its predecessor vertexes set, that is, the service can be deployed only after the services in its sPre are deployed. sPre = { sv_1, sv_2, ⋯, sv_k}, k = |sPre| is the number of vertexes which belong to sPre.
- eType is the type of hyperedge, including CH and NCH. CH is a critical hyperedge. It indicates that current hypergragh partitions will be changed if one vertex that belongs to it moves from one partition to another. NCH is a non critical hyperedge except to CH.

Definition 4: FB and TB, FB is From Block. FB (v_i, e_j) is the source partition that v_i which is connected by hyperedge e_j belongs to. TB (v_i, e_j) is the destination partition.

2.2 Hypergraph Partition for Service Deployment

The services are represented by a hypergraph-based model, and we use the properties of hypergraph partition to solve the problem of dynamic service deployment. We divide vertexes in the hypergraph into different partitions on the basis of the balance formulas and map the hypergraph partitions to the servers.

Definition 5 (Hypergraph Partition): Hypergraph Partition for Service Deployment PS = {P_1, P_2,…, P_k}(as shown in Figure 1) is a final set which includes k subsets of the vertex set V of service deployment directed hypergraph SDDG = <V, E>, where: each $P_i \in PS$ ($i \in [1,k]$) is a non-empty, pairwise-disjoint subset of V; Pw = {pw_1, pw_2,…, pw_k} is the weight set which pw_i corresponds to P_i , where pw_i is the weight of partition P_i ($i \in [1,k]$), k = |PS| is the order of the partitions. Function $F(P_i)$: P_i->S_j

(i∈[1,k],j∈[1,m]) represents the mapping relation between hypergraph partition P_i and server j. PS must meet the following conditions:

- $\sum P_i \cup P_j = V, P_i \cap P_j = \varnothing (i, j = 1,2,...,k, i \neq j)$
- $pw_i = \sum_{v_j \in P_i} CA_j$, $pw_{avg} = \sum_{p_i \in PS} pw_i / \sum RC_m$ where pw_{avg} is the average of all weight pw_i
- For each $e_i \in E(i \in [1,m])$, service i can be deployed only after the services in its source set sPre have been deployed.
- For each $v_i \in V(i \in [1,n])$, the quality of service i should be guaranteed.
- Each partition p_i is balanced only if it meet the balance formula below:

$$pw_i \leq (1+\partial) pw_{avg}, \partial > 0 \quad (1)$$

∂ is a predetermined maximum load parameters of imbalance deviation.

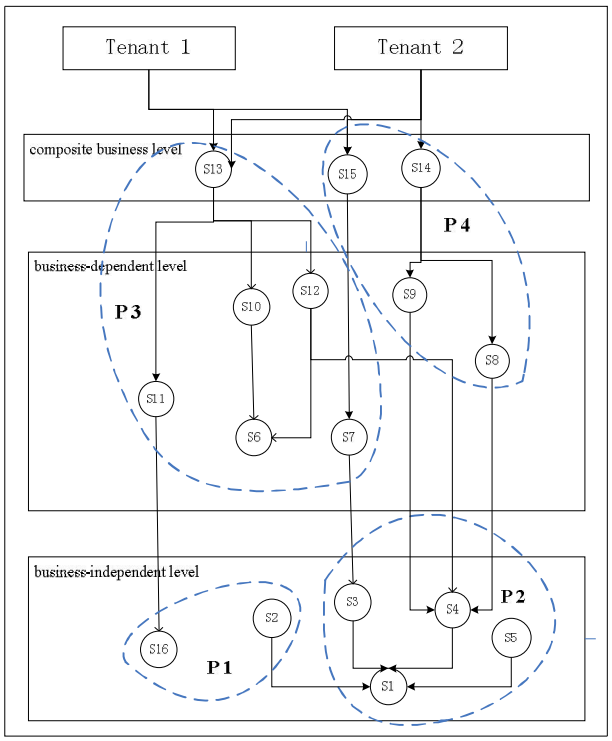

Fig. 1. There are two tenants and fifteen services in the hypergraph model. Hypergraph partition for service deployment PS = {P_1, P_2, P_3, P_4,} has four subsets.

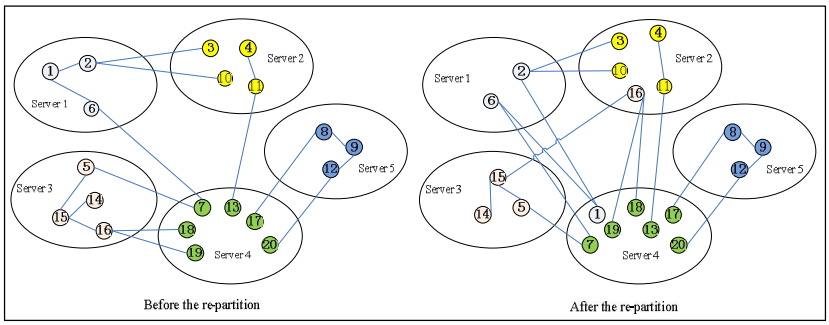

Fig. 2. The left part is the twenty service deployment before the hypergraph re-partition while the right part is the deployment after that.

3 Algorithms for Hypergraph-partition-based Service Deployment

The algorithms proposed in our paper are based on the hypergraph partition for service deployment. The main idea of the algorithms: Firstly, we initialize the service deployment directed hypergraph SDDG according to parameters. Then, we get the partitions of SDDG on the basis of balance formula. Finally, we map each partition to servers and adjust the hypergraph partitions to keep balance constantly. It mainly includes two parts: First Deployment and Re-partition.

The structure of the main algorithm described as below:

In First Deployment, we initialize the SDDG, get hypergragh partitions of service deployment and deploy the corresponding services on servers.

In Re-partition, we first get actual resource consumptions of every server from the system at regular intervals. Then we adjust the current partitions to keep load balance according to the actual resource consumptions. The resources that are considered include CPU, memory and I/O.

STEP 1 Initialization:

We initialize the service deployment directed hypergraph SDDG on the basis of parameters such as tenants and services. The hypergraph partition is based on the SDDG.

STEP 2 First Deployment:

When the so-called "first deployment", we can't get real-time resource consumptions of services, so we use the multi-tenant resource consumption model to estimate comprehensive resource consumption CA_i of each service s_i in SDDG. Then we call the function One Partition recursively until each hypergraph partition is balanced according to balance formula (1). Finally, we map each partition to a server, that is to say, we deploy the services that belong to each partition to the corresponding server.

One partition is a recursive function that belongs to First Deployment. Once called, it will check whether the current partitions are balanced. If they are balanced, function

returns. If they are not, it divides the current partitions into two parts and calculates the weight set of new partitions because estimated resource consumptions of services change with the dependence relation among them.

STEP 3 Re-partition Deployment:

The actual resource consumptions of servers change with user requests. When a server is overloaded, we should adjust current hypergraph partitions in order to re-deploy services. We can get the actual resource consumptions from server system. If the server s_i is overloaded, we select service s_i which is deployed on server s_i and has the highest sGain and move it t into the server whose load is lowest. Only when the current partitions get balance again, we stop the re-partition.

The main algorithm includes three sub-algorithms: Initialization, First Deployment and Re-partition. There are n services and m servers. The time complexity of algorithm Initialization is O(n) which has a linear relationship with the size of services. The algorithm First Deployment calls algorithm One Partition recursively and its time complexity is O(nlogn). The algorithm Re-partition is related to the size of services and servers and its complexity is O(nm). To sum up, the time complexity of whole algorithm is O(nm)+ O(nlogn)+ O(n) which is service quantity and server number related.

4 Performance Evaluation

In this section, we present the performance based on an extensive set of experiments to verify the correctness and effectiveness of our algorithms. The experiments are based on the following scene: Twenty services are deployed on five servers. The servers in our experiments are the same. The resource they provide is one unit while the limited load is 0.7 units. We consider three kinds of resource which include CPU, memory and I/O in the experiments of our paper.

Experiment 1: To verify whether the algorithm First Deployment can get the hypergraph partitions that ensure balance conditions, we list the weight of P_i after first partition of 20 services.

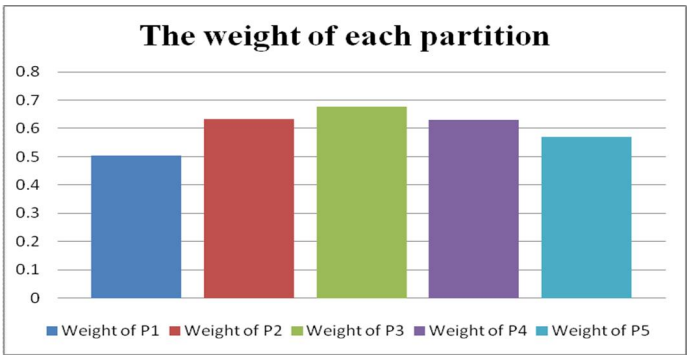

Fig. 3. We get the first partitions after the algorithm First Deployment. The weights of every P_i ($P_i \in PS$) are similar and each partition keeps balance on the basis of balance formula (1). It indicates that the algorithm can get the reasonable hypergraph partitions.

Experiment 2: The system gets the actual loads of 5 servers every day. If servers are overloaded, the system re-partitions the hypergraph according to our algorithm. Then, we adjust the deployment of services to keep server load balance. We compare the actual load of 5 servers before the re-partition with that after the re-partition.

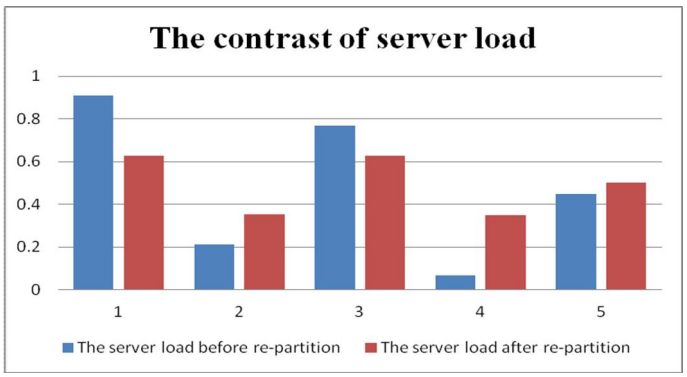

Fig. 4. The blue blocks represent the server loads before the re-partition while the red ones represent the server loads after that. The loads of 5 servers are not balanced before the re-partition deployment. Server 1 and server 3 are overloaded. After the adjustment, the loads of 5 servers are balanced again and none of them is overloaded. It indicates that our algorithm is correct and effective. Figure 2 shows the hypergraph partitions before and after the adjustment.

5 Conclusion and Future Work

In this paper, we study the dynamic deployment with the change of user requests for service-oriented multi-tenant SaaS application. We propose an approach based on hypergraph partition to solve it. The hypergraph partition is used to divide the services into several subsets while each partition is mapped to a server. If the services of every partition are satisfied to the balance formula, it means the corresponding server is not overload, that is, the servers keep balance. From the experiment results, we can conclude that our algorithm works well when we deploy services on the first time and re-deploy them when servers are overloaded.

Although the current approach is good, there are several that need to be improved in future. We plan future work to address the following considerations:

1) We consider the servers are identical and the resources that they provide are equal. In more cases, the servers are different and the resources are not equal. We will consider the differences among them.

2) Sometimes, the loads of some servers are too low, and we can combine the services deployed on them to get more value.

The two points above are not researched deeply in this paper and our future work will be developed around them.

Acknowledgment

The authors would like to acknowledge the support provided by the National High Technology Research and Development Program of China (2011AA040603, 2012AA040904), the National Key Technologies R&D Program of China (2012BAF12B07), the Natural Science Foundation of Shandong Province (ZR2009GM028, ZR2011FQ031) and Independent Innovation Foundation of Shandong University (IIFSDU)

References

1. Eric Newcomer, Greg Lomow. Understanding SOA with Web Services. Addison-Wesley (2005).
2. Thomas ERL. Service-Oriented Architecture: Concepts, Technology, and Design. New York, 2005.
3. Wei-Tek Tsai, Xin Sun, Qihong Shao, Guanqiu Qi. Two-Tier Multi-Tenancy Scaling and Load Balancing. IEEE International Conference on E-Business Engineering, Vol. 0 (2010), pp. 484-489
4. WANG De-shuai, ZHANG Yi-chua, ZHANG Bin, LIU Ying.Load Balancing Strategy for Multi-tenancy SaaS Applications Supporting Service on Demand. Journal of Northeastern U niversity(Natural Science) Vol 32, No. 3, Mar. 2011.
5. WANG Jun, ZHANG Di, WU Quan-Yuan, GUAN Yan-An. Research of a Machine-Learning Based Load Prediction Approach for the Service-oriented Computing Environment. Computer Science 2007 Vol. 34 No. 9.
6. WANG Qiong, HE Xin-hua, ZHAO Ying-kun, HU Ru-lin. Load Balancing Algorithm Based on Access Characters load Prediction. Journ al ofA cadem y ofA rm ored Force Engin eering, Oct. 2009 Vo.l 23 No. 5.
7. LIU Bin, YANG Jian, ZHAO Yu. Dynamic Cluster Configuration Strategy for Energy Conservation Based on Online Load Prediction. Computer Engineering, December 2010 Vol.36 No.24.
8. Rui Wang, Yong Zhang, Shijun Liu, Lei Wu, Xiangxu Meng. A Dependency-aware Hierarchical Service Model for SaaS and Cloud Services. 2011 IEEE International Conference on Services Computing (SCC2011), 4-9 July 2011, pp. 480-487.
9. Umit V. Catalyurek, Erik G. Boman, Karen D. Devine, Doruk Bozda, Robert T. Heaphy, Lee Ann Riesen. A repartitioning hypergraph model for dynamic load balancing. J. Parallel Distrib. Comput. 69 (2009) 711_724.
10. Thomas Kwok and Ajay Mohindra. Resource calculations with constraints, and placement of tenants and instances for multi-tenant SaaS applications. Service-Oriented Computing–ICSOC 2008 Lecture Notes in Computer Science, 2008, Volume 5364/2008, 633-648, DOI: 10.1007/978-3-540-89652-4_57.