THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 21, Number 4/2020, pp. 377–384

FOLDING APPROACH OF MULTIDIMENSIONAL ATTRIBUTES FACING TO QUALITY OF SERVICE IN CLOUD SERVICES

Jian ZHENG

Chongqing Aerospace Polytechnic, Chongqing 40021, P.R. China
E-mail: zhengjian.002@163.com

Abstract. It is a challenge to acquire different quality of service (QoS) in allocating dynamic cloud resources while meeting user's demand. To address this issue, we propose an approach of folding multi-dimensional attributes of QoS in this work. First, cloud services are encapsulated through multi-points coverage or single point coverage. Then, we fold these attributes of QoS with services function. Because the folding operation is prone to reduce the number of candidate services, both the accuracy and efficiency of the proposed approach are improved significantly. Experimental results show that the folding efficiency is augmented as QoS attributes increases. Moreover, the probability of discarded solutions that do not meet user's demand is also reduced as the scale of combination services enlarges gradually. This implies that the proposed approach promotes user's satisfaction. We find that it is not an optimal option to calculate multi-dimensional attributes through allocating weight for attributes, which indicates that the calculated approaches itself of multi-dimensional attributes is an important factor impacted to the precision and efficiency in cloud services.

Key words: quality of service, multi-dimensional attributes, cloud services.

1. INTRODUCTION

Quality of service (QoS) is an important factor impacted in the field of cloud services due to the impacted factor determines both service accuracy and efficiency. When QoS and service algorithms achieve the optimization at the same time, the efficiency of cloud services is significantly enhanced. However, these algorithms of deployed cloud services are a complex process, for instance, multi-cloud which has the ability to hide the complexity of service selection needs to introduce a cloud resource management layer [1]. The complex cloud services algorithms have a certain of negative effects for supplying QoS in cloud applications, e.g., performance, availability and reliability [2].

Studies usually assume that the applicants can identify and predict the performance of cloud resources. If the assumption is improper, the quality of cloud services receives negative effects, furthermore, the robust of service selection strategy is also reduced. The relationships between cloud resources and cloud services present several typical features, having that.

(1) Complex cloud services and dynamic cloud resources

Cloud resources that have some non-linear features show highly dynamic performance during cloud services running, so that running jobs are easy to influence the status of cloud resources. In addition, the combination of cloud services are also a complex process.

(2) Complexity of hierarchical structure

A typical process of cloud services, having that
$s1$: resources allocation.
$s2$: jobs scheduling, sub-task running.
$s3$: calculation QoS (CPU, memory, storage…).
$s4$: More scale…

Each scale, which has a structure to deal with emerging behavior, does not display chaos, i.e., each scale can keep emerging behavior orderly. Within cloud, cloud resources work together for completing cloud services. Unfortunately, cloud resources may compete each other for QoS.

Generally, the model to cloud services consists of three types of participants [3], i.e., cloud services providers, network service providers, and end users. Cloud resources serve as a basic component of cloud services, so cloud resources should be optimized to achieve the high-efficiency quality of cloud services. To optimize resources, C.T et al. [4] examine the issue of resources allocation for two (or more) multimedia service providers. To categorize cloud applications workload, Singh et al. [6] suggest to allocate resources according to common patterns before actual scheduling. In addition, dynamically cloud resources are also allowed to allocate using virtual machines [7]. Above mentioned approaches segment cloud resources using coarse-grained patterns, the approaches do not consider the differences of QoS attributes. Certainly, a extended self-tuning fuzzy control approach [8] can be also used for cloud applications, which ensures qualitative specification for applications running on cloud. Although the extended self-tuning fuzzy control approach considers the differences of QoS, but this is only used for controlling cloud applications rather than allocating cloud resources. In fact, Aymen et al. [9] design an architecture of thin client-Edge computing collaboration for cloud resources allocation, unfortunately, the architecture needs hardware while working, moreover, calculation complexity is also high.

2. RELATED WORK

Seen from a global view, cloud services are considered to be the process of acquiring cloud resources. Seen from a local view, the allocated process of cloud resources addresses a certain of complexity because of calculation QoS. QoS includes usually multi-dimensional constraints, so that QoS may exhibit several conflicts. Hence, we focus on analyzing multi-dimensional constraints to mitigate QoS attributes conflict. Usually, solved multi-dimensional constraints have two kinds of methods, one of which is that we change multi-dimensional goal function into one-dimensional goal after calculation the sum of weight to multiple QoS targets. The advantages are simple and convenient, while this is difficult to determine weight coefficient of influenced the results. The other one is that we opt for a set of the optimal non-inferior solutions to final goal constraints using heuristic methods, whose advantages need not to explore specify weight coefficient. Although heuristic approaches, such as, genetic algorithm, calculate the optimal results within a polynomial time, the complexity increases along with the problem scale. As cloud services increases to a certain scale, the existed heuristic approaches hardly achieve service demand. Hence, we did three setting of experiments to combination services using genetic algorithm. The detailed experimental description is as following.

Table 1

Execution time of genetic algorithm (same attributes)

iterations :100, population:100, rt <1000 ms			
Experiment 1 (contained candidate service = 20)		Experiment 2 (contained abstract service = 10)	
Abstract service	Time (ms)	Candidate service	Time (ms)
10	27859	20	25462
30	52987	40	48723
50	108930	60	119157
70	210581	80	230743

Table 2

Execution time of genetic algorithm

iterations :100, population:100 Experiment 3 (abstract service = 10, contained candidate service = 20)	
QoS attributes	time (ms)
rt < 1000 ms	27859
rt < 1000 ms, df > 10 M/s	59280
rt < 1000 ms, df > 10 M/s, sa > 95%	71803
rt < 1000 ms, df > 10 M/s, sa > 95%, fz > 100 M	113581
rt < 1000 ms, df > 10 M/s, sa > 95%, fz > 100 M, fa > 95%	157429

Let us assume that QoS only contains an attribute, i.e., response time, in the experiment 1and experiment 2, respectively. In experiment 1, we assume that each abstract service contains 20 candidate

services. The scale of services is enlarged by increasing the number of abstract services (linear growth), in Table 1. In experiment 2, we assume that each candidate service contains 10 abstract services. The scale of services is changed via augmenting the number of candidate services (linear growth), in Table 1. In experiment 3, we assume 10 abstract services. Each abstract service contains 20 candidate services. QoS attributes are added gradually, i.e., response time (rt), data flow (df), success accuracy (sa), file size (fz), file accuracy (fa). The results are listed in Table 2.

Analysis of executed time, the efficiency of genetic algorithm will reduce when the abstract services or candidate services increase to a certain scale. For example, the scale of abstract services is up to 70, at the same time, each abstract service contains 20 candidate services, then execution time to genetic algorithm is 210581 ms, in experiment 1, as shown Table 1. This mean that user waits for 3.5 minutes to obtain service response. On the contrary, the efficiency of heuristic approaches decreases rapidly when we enlarge QoS attributes. For example, to gain service response, user waits 2.6 minutes when QoS attributes only reaches 5, in Table 2.

Through above experimental analysis, it is a challenge to acquire different QoS in allocating dynamic cloud resources while meeting user's demand. Aiming to the problem, we propose a folding approach of multi-dimensional attributes facing to QoS. The advantages are that the non-inferior solutions are retained in the process of folding multi-dimensional attributes. Importantly, the approximated region of the optimal solutions enlarges significantly. In addition, the efficiency of cloud service is also improved, thereby saving time cost.

We summarize the main contributions of this work as follows:

(1) We propose an approach of folding multi-dimensional attributes facing to QoS, which successfully achieves to fold these attributes of QoS associated with services function while ensuring the precision and efficiency.

(2) In folding multi-dimensional attributes, we retain the non-inferior solutions, so that the approximated region of the optimal solutions is augmented.

(3) It is not an optimal option to calculate multi-dimensional attributes using weighted methods. Since weighted methods are not prone to fold services combination, calculated efficiency is not easy to guarantee.

(4) The approach itself to calculation multi-dimensional attributes is a key factor impacted in services accuracy and efficiency. A good multi-dimensional attribute method folded can be more likely to eliminate redundant attributes, thus improving service quality and reducing calculated scale.

3. METHOD

3.1. Formal definitions

To describe our method in detail, we firstly give some formal definitions.

Definition 1. Single point coverage. User has a functional demand point r. Service SW encapsulates a set of operations $SW=\{sw_i \mid i=1,2,3,\ldots\}$. If $\exists\forall sw_i \in SW$, the sw_i conforms to the conditions $sw_i \rightarrow r$. Therefore, we define that service sw_i successfully covers the functional demand point r, and denotes $sw_i.true(r)=SUCCESS$.

Definition 2. Multi-points coverage. A set of functional demand points $R=\{r_j \mid j=1,2,3,....,m\}$ and service combination $CS=\{s_k \mid k,\ u=1,2,3,\ldots..,n\}$. If $\exists\forall r_j \in R$ and $s_k \in CS \wedge s_u \in \{CS-s_k\}$ $(k \neq u)$, they are obedient to the conditions $(s_k.true(r_j)=\text{TRUE}) \Lambda (s_u.true(r_j)=\text{FAIL})$. Thus, we define that service combination CS successfully covers R, and denotes $CS.true(R)=SUCCESS$.

For all CS, if multi-points coverage succeeds, there is a kind of optimal service combination at least, so that service function folding can be accepted, and denote $S \cap_{access} R$. We use mathematical model to express a process of folding, having that

$$\begin{cases} \max(S \cap_{access} R) = G \times [1,1,1,....,1]^{\mathrm{T}} \\ s.t. Z \times G \in S \cap_{access} R \end{cases} \tag{1}$$

where, $G=[g_j]^T$ is an n-dimensional 0-1 vector. If $g_j=1$, the service sw_i covers functional demand point j, and denotes $sw_i.true(r)=SUCCESS$, otherwise, $g_j=0$. The item $Z=[z_{ij}]$ $(0<i<m, 0<j<n)$ is a 0-1 matrix of m row and n column. m, n are the number of functional demand and available resources of services combination, respectively. If $z_{ij}=1$ is true, we denote $CS.true(R)=SUCCESS$, otherwise, $z_{ij}=0$. The items $Z\times G\in S\cap_{access}R$ ensures that functional demand has a kind of services function folding at least. The $\max(S\cap_{access}R)=G\times[1,1,1,\ldots,1]^T$ ensures that a kind of service function folding is allowed to accept.

Matrix Z is used for coverage of service functional demand, in Fig. 1. CS_1, CS_2, CS_3 are structural solution vectors, respectively. $CS_1=\{0,1,0,1,0,1,0,0\}$, $CS_2=\{0,1,0,0,0,1,0,1\}$, $CS_3=\{1,0,1,0,0,0,1,0\}$. Services S_1, S_6, S_8 cover all demand points. Services S_3, S_4 cover demand points r_3, r_4. Service S_5 covers demand points r_1, r_3. Service S_7 covers demand point r_5. Service S_2 does not cover any demand points.

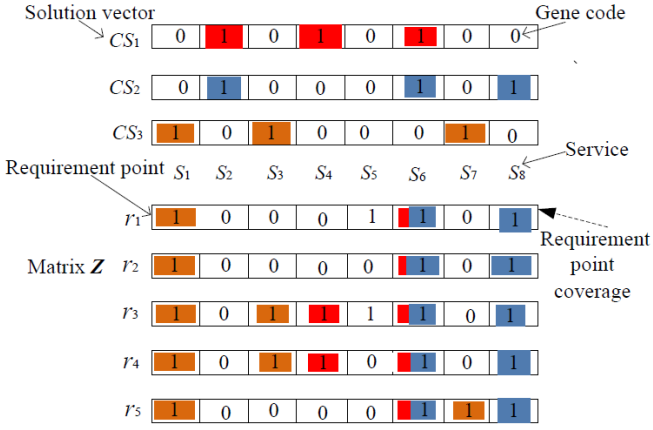


Fig. 1 – Set coverage.

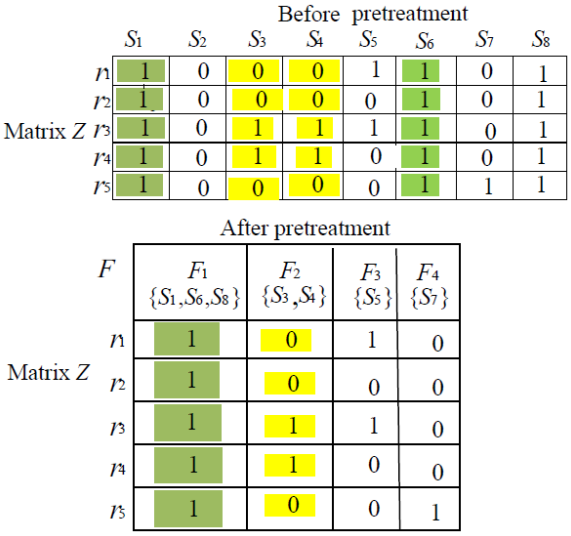


Fig. 2 – Services pretreatment.

We pretreat entire service resources S, as shown Fig. 2. Through using the cluster projection of projecting resources points onto functional demand points, services combination is implemented. If projection is an empty set, which indicates that resources points provide meaningless service to functional demand points, so the demand point is directly discarded. $F=\{F_i \mid i=1,2,3,4\}$ represents a set of services cluster, each of which covers the same functional demand points. The number of services cluster $|F|$ determines encoding length. If a cluster is accepted, gene code is marked as 1. Otherwise, gene code is marked as 0. In folding, services S_1, S_6, S_8 are clustered into a service cluster F_1. Service S_3, S_4 are clustered into another a service cluster F_2. Due to service S_2 does not cover any demand points, S_2 is directly discarded.

3.2. Algorithms

Next following, let us describe in detail several algorithms, i.e., algorithm *Crossing*(.), algorithm *Compos*(.) and *Filter*(.). These algorithms are helpful to achieve coverage of service functional demand.

(i) Algorithm *Crossing*(.) is used for crossing chromosomes. We randomly opt for a genetic crossover point to generate next generation, in Fig. 3. Given that we randomly opt for a crossover point, the results of evolved individual may be failure to achieve demand coverage. Here there are three situations of coverage, i.e., redundant coverage, incomplete coverage and uncovered coverage. Aiming to the last two situations, we introduce a composite operation *Compos*(.) into them as compensation.

(ii) Algorithm *Compos*(.), the compensation principle is that we opt for a service cluster to cover some uncovered functional points. Then, we repeatedly look for some uncovered functional points until all functional points are completely covered. Finally, we filter redundant coverage by using an algorithm *Filter*(.). If individual successfully achieves coverage demand, we product a new individual by transforming genetic code from 1 to 0. If new produced individual still satisfies original coverage demand, namely redundant gene, then the new individual is filtered. The individual of fitness function is defined, having that

$$\begin{cases} fitness(CS) = \sum_{f_i \in F} U(f_i) \\ U(f_i) = \alpha / \beta \end{cases} \tag{2}$$

where, f_i represents the gene whose code symbol is 1. $U(f_i)$ is a utilization rate of cluster, which is used to judge individual. α represents the number of gene whose code symbol is 1. β represents the number of all genes. In Fig. 1, we calculate the fitness value of CS_1, CS_2, CS_3 for the first generation chromosomes, respectively, i.e., $fitness(CS_1)=1/3+1/2\approx 0.833$, $fitness(CS_2)=2/3\approx 0.667$, $fitness(CS_3)=1/3+1/2+1\approx 1.833$. The value of CS_3 is superior to both CS_1 and CS_2, so CS_3 is opted for producing a new individual.

We present a process of decoding chromosomes, in Fig. 4. These service clusters whose gene code symbol is 1 is opt for applying, i.e., F_1, F_3. Thereafter, the service clusters are sequentially translated into chromosome codes by referring to the service code of binary. Finally, the chromosome 0101 1111 0110 0001 corresponds to service combination $\{S_1, S_6, S_8, S_5\}$ in turn.

To solve multi-dimensional attributes, we present a set of locally optimal non-inferior solution (LOPNS) algorithm. We divide local solutions into several group, then call an algorithm *Merge-Sort*(.) within group. Each group calls a algorithm G*roup-Merge*(.) to optimize local service combination. Finally, we obtain a set of globally optimal non-inferior solution (*GOPNS*) via the combination of *LOPNS*. In algorithm *Group-Merge*(los_i, los_j), the *CalQos*(.) is use to calculate QoS constraints. If *CalQos*(Δlos_n) exceeds *CalQos*(Δlos_m), we denote as *CalQos*(Δlos_n) >> *CalQos*(Δlos_m). The *LOPNS* algorithm is described in Algorithm 1.

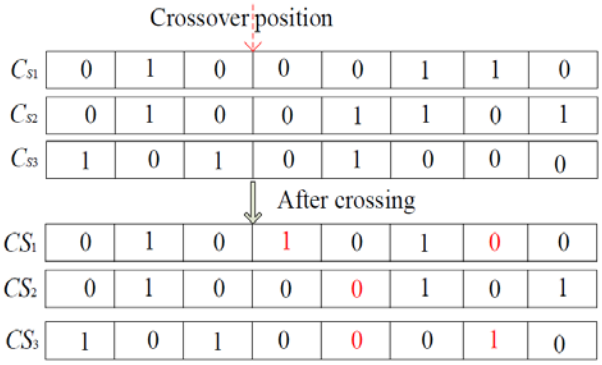


Fig. 3 – Randomly cross.

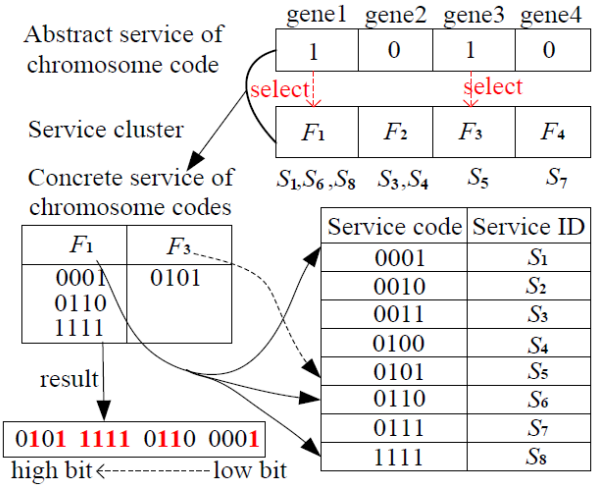


Fig. 4 – Chromosome decode.

Algorithm 1. *LOPNS*

```
INPUT:    L={los1,los2,...,losn}.          /*locally optimal solution*/
OUTPUT:   G={gos1, gos2,...,}.             /*globally optimal solution*/
Merge-Sort (L){
        WHILE( |L|/2){                     /* binary search  */
                j = 1+ |L|/2 ;             /* | L | is the number of {los1,los2,...,losn}*/
                L += Group-Merge (losi,losj);   /*call algorithm*/
        }
        IF  |L| mod 2 == 1
                L += L|L|;
        WHILE(|L| > 1) {
                Merge-Sort (LOPNS);        /* recursive call */
                |L| /= 2;
        }
        output(G);
}
```

```
Group-Merge (los_i,los_j){
    FOR(m=|los_j|; m>0; m--) {
        Δlos_m = los_j(m);                                  /*get the element m from los_j */
        flag = 0;
        FOR(n=|los_i|; n>0; n--) {
            Δlos_n=los_i(n);                                /* get the element n from los_i */
            IF CalQos (Δlos_m) >> CalQos (Δlos_n) {         /* Calculate QoS value*/
            delete element Δlos_n from the los_i;
            if los_i does not contain Δlos_m then
                    add element Δlos_m into the los_i;
            }
            IF CalQos (Δlos_m) == CalQos (Δlos_n){
                if (Δlos_m≠Δlos_n) &&(los_i does not contain Δlos_m) then
                    add element Δlos_m into the los_i;
             }
             IF CalQos (Δlos_n) >> CalQos (Δlos_m)
                flag++;
                break;
            }
        }
        IF flag == |los_i|
            add element Δlos_m into the los_i;
    }                                                       /* END-FOR */
    return (los_i);
}
```

4. RESULTS AND DISCUSSION

4.1. Experimental settings

We use 7 physical nodes in experiments. In addition, let us assume that each server is a physical node, as well as, a virtual machine is a service resource node. Each server has 5 virtual machines, i.e., service resources nodes S=35. Population scale N is considered by an empirical value.

Experimental datasets originate from the open Data Set of WS-DREAM (distributed reliability assessment mechanism for Web services). The Data Set of WS-DREAM contains 150 files, each of which contains the 100 kinds of calling information, furthermore, each calling information contains rich content, i.e., user address, response time etc. We opt for the 80 kinds of services from Data Set of WS-DREAM to verify the proposed method. Where, we consider the 5 types of QoS attributes, i.e., *response time* < 1000ms, *data flow* >100M/s, *success accuracy* > 95%, *file accuracy* > 95%, *file size* >100M.

4.2. Results and discussion

Experimental results show that execution time of our algorithm mainly spends on merge-sort and encoding/decoding gene. The results are listed Table 3. Compared with the results of both Table 1 and Table 2, our execution time is lower than them. The number of approximated solutions (NAS) is 2200 when the number of service (NS) reaches 80. Decode/encode, merge-sort time are 2200 ms and 3800 ms, respectively.

Table 3

Time cost results

Time cost			
NS	NAS	merge-sort time ($\times 10^3$ ms)	encode/decode time ($\times 10^3$ ms)
20	300	1.3	0.8
40	800	1.8	1.1
60	1300	2.5	1.3
80	2200	3.8	2.2

The distribution of approximated solutions is as shown in Fig. 5. The results show that when NS increases from 20 to 80, the percentage of discarded solution (PDS) reduces gradually. Moreover, the more approximated solutions close 100%, the better precision service is. It can be seen that service precision improves as the NS augments in turn, due to the PDS decreases gradually.

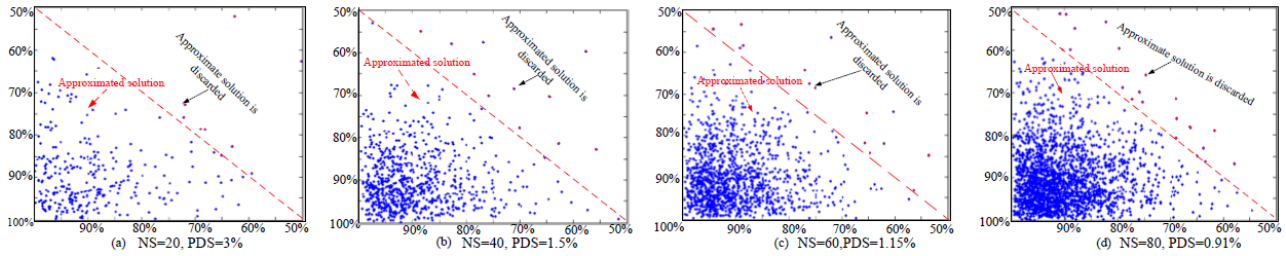
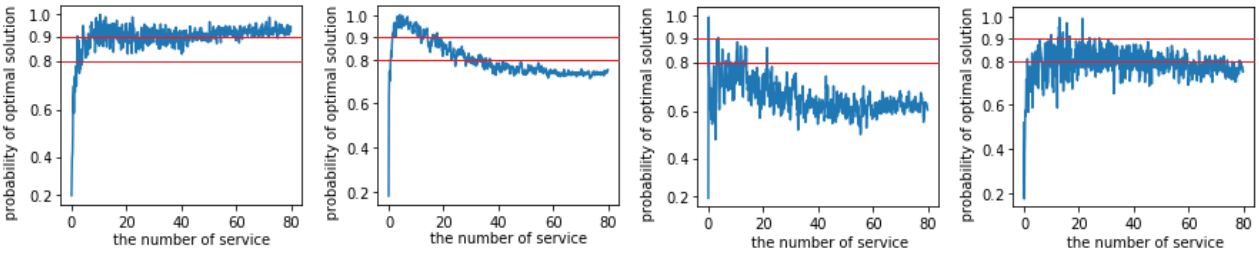


Fig. 5 – Approximated solutions distribution.

To compare with our method, we introduce three calculation ways into QoS attributes, i.e., average weighted, proportion weighted and random weighted. It should be noted that our approach calculates QoS attributes by folding attributes of QoS with services function. With same experimental conditions, we apply above three ways to replace our original calculation way in QoS attributes, respectively. To observe the optimal solutions, we provide a visual representation to the probability distributions of the optimal solutions described by all methods, as shown in Fig. 6. The results show that our method exceeds the competitive methods. The probability of the optimal solution to our method reaches 0.9, but that of competitive methods is low than 0.8, in Fig. 6. We find that it is not an optimal way to calculate multi-dimensional attributes using weighted methods. Importantly, the approach itself to calculation multi-dimensional attributes is an key factor impacted in services accuracy and efficiency.



(a) our method (b) average weighted (c) random weighted (d) proportion weighted

Fig. 6 – Optimal solutions of distribution.

5. CONCLUSION

To address this issue that user acquires different QoS in allocating dynamic cloud resources while meeting their demand. In this work, we propose an approach of folding multi-dimensional attributes facing to QoS. In terms of efficiency, due to the scale of candidate services is reduced after folding the attributes of QoS, the execution time is obviously decreased. To improve service precision, we associate with service function while folding QoS attributes. However, we may lose a little the optimal solutions in folding. In future work, we will focus on exploring the problem of losing a little the optimal solutions.

ACKNOWLEDGMENTS

The research funding is Supported by the Science and Technology Research Program of Chongqing Municipal Education Commission of China (Grant No. KJQN201903003).

REFERENCES

1. V.I. MUNTEANU, C. ŞANDRU, D. PETCU, *Multi-cloud resource management: cloud service interfacing*, Journal of Cloud Computing Advances, Systems and Applications **3**, *3*, pp.1–10*,* 2014.
2. H. CHEN, F.Z. WANG, N. HELIAN, *Entropy4cloud: Using entropy-based complexity to optimize cloud service resource management*, IEEE Transactions on Emerging Topics in Computational Intelligence*,* **2**, *1*, pp. 13–24, 2018.
3. Y. WANG, X. LIN, M. PEDRAM, *A game theoretic framework of sla-based resource allocation for competitive cloud service providers*, Proc. 6th Annual IEEE Conf. Green Technol. (GreenTech), pp. 37–43, 2014.
4. C.T. DO, Optimal resource allocation for multimedia application in single and multiple cloud computing service providers, Proc. 16th Asia-Pac. Netw. Oper. Manag. Symp. (APNOMS), pp. 1–4, 2014.
5. K.G. SRINIVASA, *Game theoretic resource allocation in cloud computing*, Proc. 5th Int. Conf. Appl. Digit. Inf. Web Technol. (ICADIWT), Bengaluru, India, pp. 36–42, 2014.
6. S. SINGH, I. CHANA, *Q-aware: Quality of service based cloud resource provisioning*, Comput. Elect. Eng., **47**, pp. 138–160, 2015.
7. Z. XIAO, W. SONG, Q. CHEN, *Dynamic resource allocation using virtual machines for cloud computing environment*, IEEE Trans. Parallel Distrib. Syst., **24**, *6*, pp. 1107–1117, 2013.
8. J. RAO, Y. WEI, J. GONG, *QoS guarantees and service differentiation for dynamic cloud applications*, IEEE Trans. Netw. Serv. Manage, **10**, *1*, pp. 43–55, 2013.
9. A. ALSAFFAR, P. HUNG, E.-N. HUH, *An architecture of thin client-edge computing collaboration for data distribution and resource allocation in cloud*, The International Arab Journal of Information Technology, **14**, *6*, pp. 842–850, 2017.

Received July 9, 2020