Efficient Coalition Formation for Web Services

Michael Shell, *Member, IEEE,* John Doe, *Fellow, OSA,* and Jane Doe, *Life Fellow, IEEE*

Abstract—Web services are loosely-coupled business applications willing to cooperate in distributed settings within different groups called communities. Communities aim to provide better visibility, efficiency, market share and total payoff. There are a number of proposed mechanisms and models on aggregating web services and making them cooperate within their communities. However, forming optimal and stable communities as coalitions to maximize individual and group efficiency and income has not been addressed yet. In this paper, we propose an efficient coalition formation mechanism using cooperative game-theoretic techniques. We propose a mechanism for community membership requests and selections of web services in the scenarios where there is interaction between one community and many web services and scenarios where web services can join multiple established communities. The ultimate objective is to develop a mechanism for web services to form stable groups allowing them to maximize their efficiency and generate near-optimal (welfare-maximizing) communities. The theoretical and extensive simulation results show that our algorithms provide web services and community owners, in real-world like environments, with applicable and near-optimal decision making mechanisms.

Index Terms—Web services; Cooperative game theory; Community of services;

1 Introduction and Related Work

Over the past years, online services have become part of many scalable business applications. The increasing reliance on web-based applications has significantly influenced the way web services are engineered. Web services provide a set of stateless software functions accessible at a network address over the web. The recent developments are shifting web services from passive and individual components to autonomous and group-based components where interaction, composition, and cooperation are the key challenges [1], [2]. The main objective is to achieve a seamless integration of business processes, applications and web services. Delivering high quality services considering the dynamic and unpredictable nature of the Internet is still a very critical and challenging issue.

The need for highly available and responsive services has called for grouping and collaborative mechanisms of loosely-coupled web services, particularly in business settings. The idea of grouping web services within communities and the way those communities are engineered so that web services can better collaborate have been proposed and investigated in [3], [4], [5]. Communities are virtual groups of web services having similar functionalities [6], [7], [8], [9], but probably different non-functional quality attributes, which form the QoS parameters. When communities are used, users send their requests to the masters of those communities, which are responsible of managing the communities, forwarding the requests to the suitable member web services and checking the credentials of those members. Communities aim to provide higher service availability and performance than what individual web services can provide. The high availability of services and the community resilience to failure are guaranteed since web services can cooperate and replace each other within the same community and since there is no single point of failure in the communities architecture.

Most of the recent work on communities of services are either user-centric and focus on user satisfaction [10] or system-centric and focus on the whole system throughput, performance and utilization. There are many contributions in distributed, grid, cluster and cloud services which are system-centric. However, in real world environments and applications, both users and service providers are self-interested agents, aiming to maximize their own profit. In those environments, both parties (users and services) will collaborate as long as they are getting more benefits and payoff.

In this direction, recently [11], [12], [13] proposed mechanisms to help users and services maximize their gain. A two-player non-cooperative game between web services and community master was introduced in [12]. In this game-theoretic model, the strategies available to a web service when facing a new community are requesting to join the community, accepting the master's invitation to join the community, or refusing the invitation to join. The set of strategies for communities are inviting the web service or refusing

- *M. Shell is with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332. E-mail: see http://www.michaelshell.org/contact.html*
- *J. Doe and J. Doe are with Anonymous University.*

the web service's join request. Based on their capacity, market share and reputation, the two players have different set of utilities over the strategy profiles of the game. The main limits of this game model are: 1) its consideration of only three quality parameters, while the other factors are simply ignored; and 2) the non-consideration of the web services already residing within the community. The game is only between the community master and the new web service, and the inputs from all the other members are simply ignored. The consideration of those inputs is a significant issue as existing web services can lose utility or payoff because of the new member, which can results in an unhealthy and unstable group. The problem comes from the fact that the existing members should collaborate with the new web services, so probably their performance as a group can suffer. Existing members may even deviate and try to join other communities if they are unsatisfied. Those considerations of forming stable and efficient coalitions are the main contributions of our paper.

In [11] a 3-way satisfaction approach for selecting web services has been proposed. In this approach, the authors proposed a web service selection process that the community masters can use. The approach considerers the efficiency of all the three involved parties, namely users, web services and communities. In this work, it is shown how the gains of these parties are coupled together using a linear optimization process. However, the optimization problem in this solution tends to optimize some parameters considering all web services regardless of their efficiency and contribution to the community's welfare. Moreover, there are no clear thresholds for accepting or rejecting new web services. The solution of the optimization problem could, for instance, suggest web services already residing within the community to increase or decrease their capacity to cover up the weakness of other parties in the system. However, a high performing web service could deviate anytime it finds itself unsatisfied within the community instead of adjusting its service parameters.

In [13] a cooperative scheme among autonomous web services based on coalitional game theory has been introduced. They have proposed an algorithm to reach individually stable coalition partition for web services in order to maximize their efficiency. The communities choose new web services on the promise that it would benefit the community without decreasing any other web service's income. In their model, the worth of community is evaluated with high emphasis on availability metric and considering price and cost values only. The community structure is based on a coordination chain, where a web service is assigned as a *primary* web service and the community task destribution method, will initially invoke the primary web service and only if the primary web service is unavailable will invoke the next backup web services as they are ordered in the coordination chain. However in cooperative models, it is preferred to have a real and active cooperative activity engaging all agents to perform the tasks more efficiently. Especially nowadays with recent advancement in cloud and hardware infrastructures availability is becoming less of an issue. So the backup web services in their model have a very low chance of getting jobs, especially the ones further in chain, which is huge waste of web services capabilities.

Contributions. In this paper, we use game theory to propose a cooperative game model for the aggregation of web services within communities. The solution concepts of our cooperative game seeks to find efficient ways of forming coalitions (teams) of web services so that they can maximize their gain and payoff, and distribute the gain in a fair way among all the web services. Achieving Fairness when the gain is distributed among the community members is the main factor to keep the coalition stable as no web service will expect to gain better by deviating for the community. In other words, the coalition is made efficient if all the members are satisfied. We first propose a representation function for communities of web services based on their QoS attributes. By using this function, we can evaluate the $worth$ of each community of web services. When facing new membership requests, a typical community master checks whether the new coalition having the old and new set of web services will keep the community stable or not. The community master will reject the membership requests if it finds out that the new coalition would be unstable, preventing any subset of web services from gaining significantly more by deviating from the community and joining other communities or forming new ones. The computation of solutions for cooperative game theory problems is combinatorial in nature and proven to be NP-complete [14], making this computation impractical in real world applications. However, using the concepts of coalition stability, we propose approximation algorithms running in polynomial time providing web services and community masters with applicable and near-optimal decision making mechanisms.

The rest of paper is organized as follows. Section 2 describes the architecture, considered parameters and some basic cooperative game theory concepts. Section 3 provides the problem statement and some numerical examples. Section 4 presents our solution model in two different scenarios. Section 5 describes our experiments and results. Finally, Section 6, concludes the paper and identifies future work.

2 PRELIMINARIES

In this section, we discuss the parameters and preliminary concepts that we use in the rest of the paper.

TABLE 1
List of web service QoS parameters.

Parameter	Definition
Availability	Probability of being available during a time frame
Reliability	Probability of successfully handling requests during a timeframe
Successability	Rate of successfully handled requests
Throughput	Average rate of handling requests
Latency	The average latency of services
Capacity	Amount of resources available
Cost	Mean service fee
Regulatory	Compliance with standards, law and rules
Security	Quality of confidentiality and non-repudiation

2.1 The Architecture

Our system consists of three main types of entities working together:

1) Web services are rational entities[1] providing services to end users. They aim to maximize their individual income by receiving enough requests from end users. In order to increase their revenue, web services seek for more tasks if they have the capacity and throughput to do so. Web services can join communities to have better efficiency by collaborating with others, to have access to higher market share, and to have opportunity of receiving a bigger task pool from end users. Throughout this paper, in our equations, we refer to web services as ws and to the set of web services hosted by a given community as C. To simply the notation, sometimes we simply write ws instead of $ws \in C$ to go through the elements ws of the set C.

2) Master Web Services or the community coordinators, are representatives of the communities of web services and responsible for their management. Communities receive requests from users and aim to host a healthy set of web services to perform the required tasks. They seek to maximize user satisfaction by having tasks accomplished according to the desired QoS. In fact, higher user satisfaction will bring more user requests and increase the market share and revenue of the community.

3) Users generate requests and try to find the best available services. User satisfaction is abstracted as function of quantity and quality of tasks accomplished by a given service.

2.2 Web Service Parameters

Web services come with different quality of service parameters. These parameters with a short description are listed in Table 1.

We adopted a real world dataset [15] which has aggregated and normalized each of these parameters to a real value between 0 and 1. Since requests are not shared among web services and are distributed among all of them inside a community, each one of them comes with a given QoS denoted by (QoS_{ws}). We assume that (QoS_{ws}) is obtained by a certain aggregation function of the parameters considered in Table 1. We use this quality output later in evaluating the community *worth* or *payoff* function.

1. The term rational is used here in the sense that web services are utility maximizers

2.3 Web Service Communities

Figure 1 represents the architecture of web service communities. The communities are basicly an abstract model of web services, they aggrigate web services and communicate with other entities such as UDDI registeries and users, using identical protocols as web services. Web services join communities to increase their utility by having a larger market share and task pool. Community cordinators or master web services are responsible for community development, managing membership requests from web services and distributing user tasks among the community members. Communitiy coordinators try to attract quality web services and keep the community as stable and productive as possible to gain better reputation and user satisfaction which results in having a higher market share for the community. The way the web services reside inside communities and how communities of web services are engineered is described comprehensivly in [3].

2.4 Cooperative Game Concepts

Cooperative game is a branch of game theory that studies strategies of self-interested entities or agents in a setting where those agents can increase their payoff by binding agreements and cooperating in groups. We let N be a set of players. Any subset S of N can form a group called *coalition*. A *coalitional game* is a pair $G = (N, v)$ where v is called a *characteristic function*, which given a coalition, is a function $v : 2^N \rightarrow \mathbb{R}$ mapping the set of players of the coalition to a real number $v(S)$, the worth of S. This number usually represents the output or payoff or again the performance of these players working together as coalition. If a coalition S is formed, then it can divide its worth, $v(S)$ in any possible way among its members. The payoff vector $x \in \mathbb{R}^S$ is the amount of payoff being distributed among the members of the coalition S. The payoff vector satisfies two conditions:

- $x_i \geq 0$ for all $i \in N$, and
- $\sum_{i \in S} x_i \leq v(S)$

The second criteria is called the *feasibility* condition, according to which, the payoff for each agent cannot be more than the coalition total gain. A payoff vector is also *efficient* if the payoff obtained by a coalition is distributed amongst the coalition members: $\sum_{i \in S} x_i = v(S)$. This definition of the characteristic function works in *transferable utility* (TU) settings,

Fig. 1. Architecture of Web Service communities

where utility (i.e., payoff) is transferable from one player to another, or in other words, players have common currency and a unit of income is worth same for all players [16].

When dealing with cooperative games, two issues need to be addressed:
1. What coalitions to form?
2. How to reward each member when a task is completed?
The following definitions help address these two issues.

Definition 1 (Shapley value) Given a cooperative game (N, v), the *Shapley value* of player i is given by[17]:

$$\phi_i(N,v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N|-|S|-1)!}{|N|!}(v(S \cup \{i\}) - v(S)) \quad (1)$$

Shapley value is a unique and fair solution concept for payoff distribution among the members of the coalition. It basically rewards members with the amount of marginal contribution they have to the coalition.

Definition 2 (Core) A payoff vector x is in the $core$ of a coalitional game (N, v) if and only if:

$$\forall S \subseteq N, \sum_{x_i \in S} x_i \geq v(S) \quad (2)$$

The core is basically a set of payoff vectors where no subset of players S' could gain more than their current payoff by deviating and making their own coalition $\sum_{i \in S'} x_i \geq v(S')$. The sum of payoffs of the players in any sub-coalition S is at least as large as the amount that these players could earn by forming a coalition by their own. In a sense, it is analogue to Nash equilibrium, except that core is about deviations from groups of entities. The core is the strongest and most popular solution concept in cooperative game theory. However, its computation is a combinatorial problem and becomes intractable as the number of players increases. The core of some real-world problem games may be empty, which means having the characteristic function of the game (N, v), there might be no possible distribution of payoff assuring stability of subgroups.

Definition 3 (Convex cooperative games) A game (N, v) with characteristic function $v(S)$ is convex if:

$$v(S) + v(T) \leq v(S \cup T) + v(S \cap T), \forall S, T \subseteq N. \quad (3)$$

According to a classic result by Shapley [18], convex games always have a non-empty core. We will use a variation of convexity condition in our algorithm to check whether our coalitions are stable.

ϵ-core

When the *core* set of a game is empty, it means no coalition of players can gain anything by deviating. An outcome would be unstable if a coalition can benefit even by a small amount from deviating, which is a strong requirement. In fact, in some situations, deviations can be costly, or players may have loyalty to their coalitions, or even it can be computationally intractable to find those small benefits. It would only make sense for a coalition to deviate if the gain from a deviation exceeds the cost of performing the deviation. *ϵ-core* relaxes the notion of the core, and only requires that no coalition would benefit significantly, or within a constant amount(ϵ) by deviating (see Equation 2).

$$\forall S \subseteq N, \sum_{x_i \in S} x_i \geq v(S) - \epsilon \quad (4)$$

Coalition Structure Formation

Coalition structure formation is the problem of finding the best partition of web services into teams. In these settings, the performance of an individual service is less important than the *social welfare* of the whole system, which is the sum of the values of all teams. Having the game (N, v), a coalition structure (CS) is *socially optimal* if CS belongs to set $\arg\max_{CS} v(CS)$ where $v(CS)$ is the sum of the values of all coalitions inside CS. $v(CS) = \sum_{C \in CS} v(C)$.

3 Problem Formulation and Modeling

In this section, we present the web service and community coordinator's intractions, the task distribution process and revenue models in web service communities.

3.1 Task destribution

As mentioned in section 2.3, communities are robust service providers with well stablished market share and reputation. By maintaining their reputation and performance, they attract end users which choose them as service providers to perform their tasks. The community master is characterizd by a request rate (R_C) from users. Each web service comes with a given QoS (QoS_{ws}) from which the throughput Tr_{ws} is excluded. This exclusion allows us to build our analysis on the particular value of Tr_{ws}. Throughput is the average rate of tasks a web service can perform per time unit. Thus, web services perform tasks with an average output quality of QoS_{ws} and a throughput rate of Tr_{ws}.

The community master uses a slightly modified *weighted fair queuing* method to distribute task among

its members. The goal is to allocate incoming tasks to web services with a rate matching throughput value of Tr_{ws}. In *weighted fair queuing* method *all* the input flow is multiplexed along different paths, however in our case if the input rate (R_C) of the community is more than summation of throughput values of web services, some of the input tasks will be queued and served with delay. Thus, the amount of tasks performed by community is $\sum_{ws \in C}(Tr_{ws})$ when $\sum_{ws} Tr_{ws} \leq R_C$. However, when the community has more web services having more total throughput value than community's request rate (R_C) the *weighted fair queuing* algorithm assigns a weighted task rate of $R_C \times \frac{Tr_{ws}}{\sum_{ws} Tr_{ws}}$ for each web service (ws) and the total rate of tasks being performed is R_C, the community's receiving request rate.

While distributing tasks, the community master can verify the performance, throughput and quality of tasks being performed by web services. It can recognize if web services are capable of doing the amount of tasks they advertised. If for any reason there is a decline in quality metrics or throughput, the master community will announce the new parameters and community masters and members can consider those values as benchmark for future performance calculations and also penalize them. Therefor its easy for the system to encourage players to be in some sense incentive compatible in the way that they would profit best by truthfully revealing their capabilities. Also it's important to be dynamic enough to consider web services which may have their quality metrics degraded or even improved over time for any reason and be able to adjust the community with new parameters.

3.2 Community Revenue

The communities and web services earn revenue by performing tasks. The total gain is function of quality (QoS_{ws}) and quantity(Tr_{ws}) of tasks being performed. As mentioned in section 2.2, QoS_{ws} is obtained by a certain aggregation function of the parameters considered in Table 1. We have adopted a linear equal weigth average over all QoS parameteres listed in table 1 excluding the $Throuput$ and $Cost$ parameters. A community has the option to weigh specific QoS patameters more or less depending on the expectations of their clients.

The maximum potential output $(PO(C))$ of a community is aggregation of number of tasks, times their quality, for each web service participating as a member of the community:

$$PO(C) = \sum_{ws \in C} (T_{ws} \times QoS_{ws}) \tag{5}$$

If the summation of throughput (Tr_{ws}) values of community members exceeds the input task rate of the community (R_C) the community cannot perform at its maximum potential. It means community has more web services than it needs for performing input task load. Therefore, in these cases, the actual output has to be normalized to the ammount of tasks being performed.

$$Out(C) = \begin{cases} PO(C) & \text{if } \sum_{ws} Th_{ws} \leq R_C \\ PO(C) \times \frac{R_C}{\sum_{ws} Th_{ws}} & \text{if } \sum_{ws} Th_{ws} > R_C \end{cases} \tag{6}$$

The revenue function of the web service community, is a linear function of $Out(C)$ with a positive constant multiplier.

TABLE 2
Three web services of example 1

WS	QoS_{ws}	Th_{ws}	$Th_{ws} \times QoS_{ws}$
1	0.8	4	3.2
2	0.8	5	4.0
3	0.8	3	2.4

Community	Worth	Community	Worth
$\{1\}$	3.2	$\{1,2\}$	7.2
$\{2\}$	4.0	$\{1,3\}$	6.8
$\{3\}$	2.4	$\{2,3\}$	6.4
$\{1,2,3\}$	8.0		
Community R_C: 12			

TABLE 3
Three web services of example 2

WS	QoS_{ws}	Th_{ws}	$Th_{ws} \times QoS_{ws}$
1	0.8	5	4.0
2	0.7	6	4.2
3	0.7	4	2.8

Community	Worth	Community	Worth
$\{1\}$	4.0	$\{1,2\}$	7.4
$\{2\}$	4.2	$\{1,3\}$	6.8
$\{3\}$	2.8	$\{2,3\}$	7.0
$\{1,2,3\}$	7.3		
Community R_C: 10			

3.3 Case Study

In this section, we analyse a few numerical examples and discuss the motivation of web services and community intractions and the strategies they can adopt and the revenue they can earn adopting different strategies.

We present an example with a community with R_C value of 10, and three web services each having different QoS_{ws} and Th_{ws} values as listed in table 2. The worth of a community is calculated based on $Out(C)$ equation (6) which is the amount of output being generated by the community. The first table in figure 2 lists the web services with their aggregated QoS_{ws} parameters, their task input rate while working alone,

TABLE 4
Three web services of example 3

WS	QoS_{ws}	Th_{ws}	*Input Task Rate*
1	0.8	10	5
2	0.8	20	5
3	0.8	30	5

Community	Worth	Community	Worth
$\{C_1\}$	0	$\{C_2\}$	0
$\{C_1, ws_1\}$	8	$\{C_2, ws_1\}$	8
$\{C_1, ws_2\}$	16	$\{C_2, ws_2\}$	16
$\{C_1, ws_3\}$	16	$\{C_2, ws_3\}$	24
$\{C_1, ws_1, ws_2\}$	16	$\{C_2, ws_1, ws_2\}$	24
$\{C_1, ws_1, ws_3\}$	16	$\{C_2, ws_1, ws_3\}$	32
$\{C_1, ws_2, ws_3\}$	16	$\{C_2, ws_2, ws_3\}$	32
$\{C_1, ws_1, ws_2, ws_3\}$	16	$\{C_2, ws_1, ws_2, ws_3\}$	32
$\{C_1, C_2, ...\}$	0	$\{ws_1\}$	6.8
$\{ws_2\}$	4.2	$\{ws_3\}$	6.8
Community R_{C_1}: 20			
Community R_{C_2}: 40			

and also their Th_{ws} throughput value. The second table shows all the possible communities and their worth. The values suggest, communities having more web services have better gain and output. However community tries to distribute the gain between web services. Sometimes it is impossible to share the gain between all web services in a way no subset of them would individually gain more if they had their own group. In this example, the value community of ws_1 and ws_2 is 7.2, with ws_3 joining the community the value of community increases to 8.0, however there is no way to distribute the value among web services, where ws_1 and ws_2 can earn 7.2, and ws_3 earn at least 2.4, which they could earn before joining the community. This makes the group unstable. In the example of table 3, we even have situations where a web service (ws_3) joining a community ($\{ws_1, ws_2\}$) decreases the value of community. The reason is, the community is already full and all tasks are almost being distributed and new community with bad quality can degrade the average quality of tasks being done by the community. In both examples, the request of joining of web service ws_3 should be rejected by the community.

Now we consider another example, in which we have different communities with different market share, R_C values. Web services also have a small share of market independently, providing them with a small task pull. In these kind of scenarios, the solution considers individual maximization of payoff and also the total worth of all communities which represents the $socialwelfare$. In this example the most efficient partition of web services is earned by having two coalitions of $\{C_1, ws_2\}$ and $\{C_2, ws_1, ws_3\}$, which yields a total value of $32 + 16 = 48$. In these types of scenarios, the goal is to reach stability, adopting a distributed approach where all players have the power of choice on the decision of whether or not they join a coalition. The communities usually start the game having some established members, encountering new web services, the communities may exchange web services and new web services would join them having at least one player gaining utility, without hurting any other participant. In this example if we initially having two coalitions of $\{C_1, ws_2\}$ and $\{C_2, ws_1\}$ and a ws_3 as new web service, ws_3 joining C_1 would hurt at least itself or ws_2, however ws_3 joining C_2 would not hurt any participants and ws_3 would earn more within the community and the community will have enough web services performing the incoming tasks from users.

Fig. 2. Web Services and A Grand Community

4 Web Service Cooperative Games

In this section, we present different web service community models and focus on the problem of how both web services and community masters as rational entities would adopt strategies to maximize their payoff.

4.1 Web Services and One Community

In this scenario, we assume the existence of a typical community managed by its master, and web services need to join it to be able to get requests from the master. The community master is characterized by a requests rate (R_C) from users. Each web service comes with a given QoS (QoS_{ws}). The worth of a community $v(C)$ is set to Out(C) based on equation 6.

As mentioned in previous section, the worth and output of a community of web services is a function of the throughput and provided QoS. If the throughput rate is more than the master's input request rate, it means the web services inside the community are capable of serving more requests than the demand. Considering this factor, the valuation function is designed to balance the output performance so that it matches the exact throughput rate and QoS the web service can provide within the particular community.

In this first scenario, we only consider one grand coalition and analyze the system from the point of view of one single master web service and a collection of web services. The master web service decides which members can join the community and distributes the requests and income among its community members (see Figure 2).

The membership decision is made based on throughput and *QoS* of the considered web service. The goal is to have quality web services in the community so it stays stable and no other web services would have incentives to deviate and leave the coalition C. Therefore, a basic method would be to check the core of the coalition C considering all the

current community members (all web services already residing within the community) and the new web service. This algorithm uses the *Shapley value* distribution method as described in Equation 1 to distribute the gain of $v(C)$ among all the members and then checks if the *Shapley value* payoff vector for this community having the characteristic function $v(C)$ is in the *core*. In the *Shapley value* payoff vector, the payoff for each web service ws_i is calculated based on its marginal contribution $v(C \cup i) - v(C)$ over all the possible different permutations in which the coalition can be formed, which makes the payoff distribution fair. Because of going through all the possible permutations of subsets of N, the nature of the *Shapley value* is combinatorial, which makes it impractical to use as the size of our coalitions grows. However, it is proven that in convex games, the *Shapley value* lies in the core [19], [16]. Thus, if the *Core* is non-empty, the payoff vector is a member of the *Core*. The following proposition is important to make our algorithm tractable.

Proposition 1: A game with a characteristic function v is convex if and only if for all S, T, and i where $S \subseteq T \subseteq N \backslash \{i\}, \forall i \in N$,

$$v(S \cup \{i\}) - v(S) \leq v(T \cup \{i\}) - v(T) \qquad (7)$$

Proof: We first prove the "only if" direction:

1. "only if" direction:

Assume:

$$v(S \cup \{i\}) - v(S) \leq v(T \cup \{i\}) - v(T)$$
$$\rightarrow v(S \cup \{i\}) + v(T) \leq v(T \cup \{i\}) + v(S)$$

Considering $S \subseteq T$:

$$S \cup \{i\} = (S \cup \{i\}) \cup T$$
$$T = (S \cup \{i\}) \cap T$$

By setting $A = S \cup \{i\}$ and $B = T$ we have:

$$v(S \cup \{i\}) + v(T) \leq v(T \cup \{i\}) + v(S)$$
$$\rightarrow v(S \cup \{i\}) + v(T) \leq$$
$$v((S \cup \{i\}) \cup T) + v((S \cup \{i\}) \cap T)$$
$$\rightarrow v(A) + v(B) \leq v(A \cup B) + v(A \cap B)$$

Consequently, the game is convex.

2. "if" direction:

Assume the game is convex. Thus, for all $A, B \subset N$, we have:

$$v(A) - v(A \cap B) \leq v(A \cup B) - v(B)$$

By setting $S \cup \{i\} = A$ and $T = B$ where $S \subseteq T$:

$$v(S \cup \{i\}) - v((S \cup \{i\}) \cap T) \leq v(T \cup (S \cup \{i\})) - v(T)$$
$$\rightarrow v(S \cup \{i\}) - v(S) \leq v(T \cup \{i\}) - v(T)$$

□

Thus, in order to keep the characteristic function convex, new web services should have more marginal contribution as the coalition size grows.

Fig. 3. Web Services and Many Communities

Our algorithm works as follows. We have an established community with a master and some member web services already residing in the community. A web service would send a join request to the community. The ideal solution would be analyzing the *core* or *ϵ-core* stability of the group having this new member. As the normal core membership algorithm is computationally intractable, we exploit Proposition 1 and Equation 7 to check the convexity of our game having characteristic function where the new member is added. In the equation, we set C to be our community members (C) before having the new web service. We assign i to the new web service, and then verify the equation for S, setting $S = T/W1$ where $W1$ is the set of all possible subsets of the set N having the size 1. We can relax the equation a bit by adding a constant ϵ to the left side of the equation. We call this method *Depth-1 Convex-Checker* algorithm. If the equation is satisfied for all $W1$, we let the new web service join our community, since the web service will contribute positively enough to make our new community stable. Since only subsets of size 1 are checked, the following Proposition holds.

Proposition 2: The run time complexity of Depth-1 Convex-Checker algorithm is $O(n)$.

By this result, we obtain a significant reduction from $O(2^n)$, which is the complexity of checking all possible subsets of N. In our second method, we use the same algorithm, but this time we set $W2$ to be the set of all possible subsets of size two and one of the community C. We call this method *Depth-2 Convex-Checker* and its run time complexity is still linear:

Proposition 3: The run time complexity of Depth-2 Convex-Checker algorithm is $O(n^2)$.

It is possible to develop an anytime algorithm by continuing the verification of this condition against all subsets of size 3, 4, etc. until the algorithm gets interrupted.

4.2 Web Services and Many Communities

In this scenario, we consider multiple communities managed by multiple master web services, each of which is providing independent request pools (see Figure 3). Identical to the first scenario, master web services form coalitions with web services. We use coalition structure formation methods to partition web services into non-empty disjoint coalition structures. As mentioned in Section 2.4, the used algorithms [20], [19], [21] try to solve key fundamental problems of what coalitions to form, and how to divide the payoffs among the collaborators.

In coalition-formation games, formation of the coalitions is the most important aspect. The solutions focus on maximizing the social welfare. For any coalition structure π, let $v_{cs}(\pi)$ denote the total worth

$\sum_{C\in\pi} v(C)$, which represents the *social welfare*. The solution concepts in this area deal with finding the maximum value for the social welfare over all the possible coalition structures π. There are $centralized$ algorithms for this end, but these approaches are generally NP-complete. The reason is that the number of all possible partitions of the set N grows exponentially with the number of players in N, and the centralized algorithms need to iterate through all these partitions. In our model, we propose using a distributed algorithm where each community master and web service can be a decision maker and decide for its own good. The aim is to find less complex and distributed algorithms for forming web services coalitions[22], [23], [24]. The distributed merge-and-split algorithm in [22] suits our application very well. It keeps splitting and merging coalitions to partitions which are preferred by all the players inside those coalitions.

This merge-and-split algorithm is designed to be adaptable to different applications. One major ingredient to use such an algorithm is a preference relation or well-defined orders proper for comparing collections of different coalition partitions of the same set of players. Having two partition sets of players, namely $P = P_1, ..., P_k$ and $Q = Q_1, ...Q_l$, one example would be to use the social welfare comparison $\sum_{i=1}^{k} v(P_i) > \sum_{j=1}^{l} v(Q_j)$. For our scenario, we use *Pareto order* comparison, which is an individual-value order appropriate for our self-interest web services. In the Pareto order, an allocation or partition P is preferred over another Q if at least one player improves its payoff in the new allocation and all the other players still maintain their payoff ($p_i \geq q_i$ with at least one element $p_i > q_i$).

The valuation function $v(C)$ for this scenario is the same as *"Web Services and One Community"* scenario. However, in order to prevent master web services joining the same community, we set $v(C) = 0$ when C has either none, or more than one master web service as member.

In this scenario, as new web services are discovered and get ready to join communities, our algorithm keeps merging and splitting partitions based on the preference function. The decision to merge or split is based on the fact that all players must benefit. The new web services will merge with communities if all the players are able to improve their individual payoff, and some web services may split from old communities, if splitting does not decrease the payoff of any web service of the community. According to [25], this sequence of merging and splitting will converge to a final partition, where web services cannot improve their payoff. More details of this algorithm and analysis of generic solutions on coalition formation games are described in [22].

4.3 Taxtation, Subsiding and Community Stability

We discuessed $core$ as one the preminent solution concepts in cooperative games. Working together, completing tasks and generating revenue, agents need to distribute the gain in a way no agents would gain more by having their own group. However, in most cases the core of a game is empty so we introduced the *ϵ-core* concept, where agents would only earn a minimal ammount of ϵ by deviating. Comuunity stability is an attractice property for communities, also communities would benefit by having slightly more web services then having the exact number of web services satisfying task rate cap. One reason is there is always possibility that agents may leave the community or they may under perform from the quality values they were initially performing with.

One way for communities to ensure community stability could be applying a tax, ϵ ammount of cost for web services changing communities which would make deviation a costly act. However this would require all community coordinators to agree on same ammount of taxtation, being governed by some external entities or web services would join communities with lesser ammount of tax. Another viable solution in our scenario is to stabilize the game using external subsidies. The reason a game is not stable is that the community is not making enough revenue to allocate the players with enough gain. A community master can subside its community with a constant coefficient value of λ. Rewarding a community with high number of quality participants with $\lambda v(C)$, where $\lambda \leq 1$. Obviously with a big number of λ it is always possible to stabilize the community. However, this can be a costly act for community masters so they are interested in the minimum subside value of λ making the game stable. Subsiding or taxting in order to reduce the bargaining power of sub coalitions are called $taxtation$[26] methods.

5 Experimental Results and Analysis

In this section, we discuss the experiments we performed for our scenarios to validate the applicability and performance of our proposed methods in realistic environments. We created a pool of web services and populated most of their *QoS* parameters from a real world web service dataset [15]. We implemented the simulations using Java and executed the simulations on an Intel Xeon X3450 machine with 6GBs of memory. For other parameters missing from the dataset, we used normal random distribution with parameters estimated using the method of maximum likelihood.

One of the key criteria reflecting the performance of web service coalitions is the user satisfaction. User satisfaction can be measured in terms of quality and quantity of requests (or tasks) successfully answered by the communities. We initiate the communities with

Fig. 4. Part (a): Cumulative number of requests successfully done. Part (b): Average QoS of requests performed.

Fig. 5. Analysis of ϵ-*core* set non-emptiness, for different values of ϵ

few web services, then let rejecting and accepting random web services go for a short number of iterations. After that, we start the request distribution for the communities and let them allocate requests among member web services. Thereafter, we measure the average output performance of tasks in communities following different methods.

Figure 4 depicts the results of optimal ϵ-*core*, *Depth-1 Convex-Checker*, *Depth-2 Convex-Checker*, *3-Way Satisfaction* [11], and *2 Player Non-Cooperative* [12] methods in *one grand community with many web services* scenario.

For the *optimal core* method we have used the well known ϵ-*core* method as the taxtation method to relax the core condition to help communities, attract web services. We have assigned ϵ to 15% of total community worth, $\epsilon = 0.15 \times v(C)$, which allows subsets of the coalition to gain maximum 15% of $v(C)$. In the *optimal* ϵ-*core* method, we capped the coalition size to 25 web services, since the method is computationally intractable as number of web services increase and anything more than that would make it impractical to run in our simulations. In the other methods, there were no cap on size of the community and we had communities of size 60 web services at some points. The results show that our *depth-2 convex checker* method is performing better compared to the other methods and its performance is close to optimal ϵ-*core* method. Our *depth-1 convex checker* and the *3-Way Satisfaction* method, are also performing well.

As mentioned in Section 2, the concept of *core*, assumes no coalition of players can gain anything by deviating, which is a fairly strong requirement, and that is why the notion of ϵ-*core* was introduced. Least-Core $e(G)$ of a game G, is the minimum amount of ϵ so that the core is not empty. We evaluated the non-emptiness of ϵ-*core* set using the valuation function and a set of web services. We picked random number of web services from the dataset and formed around 10,000 random coalitions consisting of 3 to 26 web services. We choose 26 as the maximum number of members in our coalition since it is computationally very complex for larger coalitions to verify whether ϵ-*core* set is empty or not. Also instead of considering ϵ amount of constant deviation in ϵ-*core* definition (Equation 4), we similarly defined *relative* ϵ-*core* concept where no coalition would benefit more than $\epsilon \times v(C)$ by deviating. We set ϵ between 0 and 1 and verify the *relative* ϵ-*core* set non-emptiness. The results in Figure 5 illustrates that almost 10% of our random web service coalitions have non-empty *core* solution and ϵ-*core* solution is *always* non-empty when we let agents gain only 30% more of $v(C)$ by deviating.

Fig. 6. Analysis of community subsiding coefficient λ on average community size (a), cost (b), number of tasks performed (c), and average quality of service of tasks performed (d).

One of the properties of coalition structure formation algorithms in our second scenario is that they partition web services with low throughput rate so that they usually join coalitions with less request rate. Since the characteristic function $v(C)$ and the fair Shapely payoff vector is proportional to web services' contribution, the web services with small contribution will get paid much less in communities having web services with high throughput. On the other hand, according to the valuation function $v(C)$, web services with high throughput will not contribute well to communities with low amount of user requests (low market share). The strong web services are likely to deviate from weak coalitions, joining a stronger one, which makes the initial coalition unstable.

As mentioned in section 4.3 in order to help community stability, is to subside the community by a relative coefficient (λ) so the value of $\lambda v(C)$ is devided among the community members. We have analysed the effect of subsiding and the cost it incurs to our web service communities. Figure 6 shows the results. In this expirement we have set a community with input task rate R_C of 100 and having abstract web services throuput rate T_r value of a random distribution with average 10 tasks per iteration. Part (a) shows the community size increases in a linear fashion as (λ) increases. However the cost is having a slight exponential growth rate since, not only (λ) increases also the size of the community is also increasing slowly. Therefore subsiding can be costly for larger number of λ value. Part (c) of figure 6 depicts the number of tasks done by the community per iteration. It is obvoius with λ value of 1.3 which is 30% of community valuation, the number of tasks done almost reaches the input task rate cap of 100 tasks per iteration. The average quality of tasks also has a slight increase since the community will be able to afford better and more web services to join the community. These results proves with quality metrics our agents as web services have and their collabrative worth, the increase rate of subsiding more 1.3 is not very effective and is costly to perform.

In previous experiment, we consider the scenario where all web services are stable will not leave the group and will fulfill their promised quality metrics for a good period of time. However, in real world scenarios of web services, this is not always the case. This is the reason why community would be interested in subsiding a community of larger group web services, paying a few more web services in order to keep the

Fig. 7. Analysis of community subsiding coefficient λ having web service different stability levels of τ on average community size, number of tasks performed, and average quality of service, and average cost/income of communities.

Fig. 8. Part (a): Cumulative number of tasks succesfully done. Part (b): Average QoS of tasks performed.

Fig. 9. A comparison between our community model and the high availability community model, on communities of size 4,5,6 only. Part (a): Cumulative number of tasks succesfully done. Part (b): Average QoS of tasks performed. (c) Average Community Service Availability

group reliable from end user point of view. In our next scenario, we have introduced the new instability variable τ from 0 to 1, 0 meaning web services having no instability issues and will perform as they claimed until the end of experiment and 1 meaning very unstable web services which will stop functioning on the first iteration of community distributing tasks. Figure 7 illustrates the results of our experiment having web services with average instability values of 0 to 0.5 and having relative subside value λ if 1,1.3,1.6 and 2. The *Cost / Income* charts on right column, shows having subside value of 1.3 incurs the least cost and increases the community income significantly. Subsiding values of 1.6 and 2 , incur a lot of cost to community and only slightly increase the community revenue. Also the role of subsiding is much more obvious when we have unstable web services. In scenarios where web services are 100% stable, subsiding cost will hardly be compensated by community revenue.

In Figure 8, we compare our *Web Services and many Communities* scenario with a method which ignores QoS parameters and forms coalitions by allowing web services to join only if they have enough requests for themselves. In other words, web services can join a community when the request rate is less than the throughput of all the member web services. We name this method *Random Formation* and use it as a benchmark for our QoS-aware coalition formation process. As the results illustrate, our method forms better coalitions of web services improving performance and satisfaction for both web services and coalitions.

Finally, in our last experiment, we compare our work with the work in [13] which we call it *High Availability Coalition* method, because of nature of their community valuation function, which focuses on community availability as main consideration. Their community formation model is very different than ours, however we have been very careful to make the experiment environment as fair and similar as possible. We limited our maximum community size by 5 in order to have communities with almost the same size. In their method, they have used web services as backups rather than active collaborative players, and they only get task when the first web service in an ordered chain fails to perform the task. However with recent advancement in cloud and hardware infrastructures availability is less of an issue for web services, and web services are highly available. Part(a) of figure 9 shows our method performs almost performs tasks successfully with rate of three times more than *High Availability Coalition* method because of real cooperative nature and task distribution of our algorithm. This result shows using web services as backups, and not as real collabrative players is a huge waste of web service capability since they have very low chance of getting jobs and its the primary web service (the first in their coordination chain) which does most of the work. The average quality of service of tasks performed is also better since our method considers all quality of service metrics mentioned in table 1. Part(c) shows the availability of community from end user point of view. The *High Availability Coalition* method almost has 100% uptime since they use web services as backups, so the chance of job failing reduces significantly as community members increase. In our method we have more chance of fail for each web service, however with some subsiding, and hiring a few more web service they can compensate the low chance of failure of web services in our community.

6 Conclusion

In this paper, we proposed a cooperative game theory-based model for the aggregation of web services within communities. The goal of our services is to maximize efficiency by collaborating and forming stable coalitions. Our method considers stability and fairness for all web services within a community and offers an applicable mechanism for membership requests and selection of web services. The ultimate goal is to increase revenue by improving user satisfaction, which comes from the ability to perform more tasks with high quality. Simulation results show that our, polynomial in complexity, approximation algorithms provide web services and community owners with applicable and near-optimal decision making mechanisms.

As future work, we would like to perform more analytical and theoretical analysis on the convexity condition and also minimal ϵ values in ϵ-*core* solution concepts based on the characteristic function in web service applications. From web service perspective, the work can be extended to consider web service compositions where a group of web services having

different set of skills cooperate to perform composite tasks. Also bargaining theory from cooperating game theory concepts can be used to help web services resolve the instability and unfairness issues by side payments.

ACKNOWLEDGMENTS

The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...The authors would like to thank...

REFERENCES

[1] P. Rodriguez-Mier, "Automatic web service composition with a heuristic-based search algorithm," in *IEEE ICWS*, 2011, pp. 81–88.

[2] K. Benouaret, D. Benslimane, A. Hadjali, and M. Barhamgi, "Top-k web service compositions using fuzzy dominance relationship," in *IEEE SCC*, 2011, pp. 144–151.

[3] Z. Maamar, S. Subramanian, P. Thiran, D. Benslimane, and J. Bentahar, "An approach to engineer communities of web services: Concepts, architecture, operation, and deployment," *IJEBR*, vol. 5, no. 4, pp. 1–21, 2009.

[4] B. Benatallah, Q. Z. Sheng, and M. Dumas, "The self-serv environment for web services composition," *IEEE Internet Computing*, vol. 7, no. 1, pp. 40–48, 2003.

[5] S. Rosario, A. Benveniste, S. Haar, and C. Jard, "Probabilistic qos and soft contracts for transaction-based web services orchestrations," *IEEE Trans. Serv. Comput.*, vol. 1, no. 4, pp. 187–200, Oct. 2008.

[6] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality driven web services composition," in *Proceedings of the 12th international conference on World Wide Web*, ser. WWW '03. New York, NY, USA: ACM, 2003, pp. 411–421. [Online]. Available: http://doi.acm.org/10.1145/775152.775211

[7] H.-Y. Paik, B. Benatallah, and F. Toumani, "Toward self-organizing service communities," *Trans. Sys. Man Cyber. Part A*, vol. 35, no. 3, pp. 408–419, May 2005. [Online]. Available: http://dx.doi.org/10.1109/TSMCA.2004.846402

[8] B. Medjahed, "A dynamic foundational architecture for semantic web services," *Distributed And Parallel Databases, an International Journal*, vol. 17, pp. 179–206, 2005.

[9] Z. Maamar, Q. Z. sheng, and D. Benslimane, "Sustaining Web services High-Availability Using Communities." in *hird IEEE International Conference on Availability, Reliability and Security (IEEE ARES 2008)*, IEEE, Ed., Mar. 2008, pp. 834–841. [Online]. Available: http://liris.cnrs.fr/publis/?id=3375

[10] B. N. Chun and D. E. Culler, "User-centric performance analysis of market-based cluster batch schedulers," in *The 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002, p. 30.

[11] E. Lim, P. Thiran, Z. Maamar, and J. Bentahar, "On the analysis of satisfaction for web services selection," in *IEEE SCC*, 2012, pp. 122–129.

[12] B. Khosravifar, M. Alishahi, J. Bentahar, and P. Thiran, "A game theoretic approach for analyzing the efficiency of web services in collaborative networks," in *IEEE SCC*, 2011, pp. 168–175.

[13] A. Liu, Q. Li, L. Huang, S. Ying, and M. Xiao, "Coalitional game for community-based autonomous web services cooperation," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, 2012.

[14] X. Deng and Q. Fang, "Algorithmic cooperative game theory," *Pareto Optimality, Game Theory And Equilibria. Springer Optimization and Its Applications*, vol. 17, pp. 159–185, 2008.

[15] E. Al-Masri and Q. H. Mahmoud, "Discovering the best web service: A neural network-based solution," in *SMC*, 2009, pp. 4250–4255.

[16] R. Myerson, *Game theory: analysis of conflict*. Harvard University Press, 1991.

[17] L. S. Shapley, "A value for n-person games," *In Contributions to the Theory of Games (Annals of Mathematical Studies)*, vol. 2, pp. 307–317, 1953.

[18] ——, "Cores of convex games," *International Journal of Game Theory*, vol. 1, pp. 11–26, 1971.

[19] G. Greco, E. Malizia, L. Palopoli, and F. Scarcello, "On the complexity of the core over coalition structures," in *IJCAI*, 2011, pp. 216–221.

[20] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohm, "Coalition structure generation with worst case guarantees," *Artificial Intelligence*, vol. 111, no. 12, pp. 209 – 238, 1999.

[21] T. Rahwan, T. P. Michalak, and N. R. Jennings, "Minimum search to establish worst-case guarantees in coalition structure generation," in *IJCAI*, 2011, pp. 338–343.

[22] K. R. Apt and A. Witzel, "A generic approach to coalition formation," *IGTR*, vol. 11, no. 3, pp. 347–367, 2009.

[23] T. Dieckmann and U. Schwalbe, "Dynamic coalition formation and the core," *Journal of Economic Behavior and Organization*, 2002.

[24] D. Ray, *A Game-Theoretic Perspective on Coalition Formation*. OUP Oxford, 2007.

[25] K. R. Apt and T. Radzik, "Stable partitions in coalitional games," *CoRR*, vol. abs/cs/0605132, 2006.

[26] Y. Zick, M. Polukarov, and N. R. Jennings, "Taxation and stability in cooperative games," in *Proc. 12th Int. Conf on Autonomous Agents and Multi-Agent Systems*, 2013, pp. 523–530. [Online]. Available: http://eprints.soton.ac.uk/346856/

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.