Efficient Coalition Formation for Web Services

Ehsan Khosrowshahi Asl, Jamal Bentahar
Faculty of Engineeing and Computer Science
Concordia University
Montreal, Canada
e_khosr@encs.concordia.ca, bentahar@ciise.concordia.ca

Hadi Otrok, Rabeb Mizouni
Department of Electrical and Computer Engineering
Khalifa University
Dubai, UAE
hadi.otrak, rabeb.mizouni@kustar:ac:ae

***Abstract*—Web services are loosely-coupled business applications and willing to cooperate in distributed settings within different groups called communities. Communities aim to provide better efficiency, market share and total payoff. There are a number of proposed mechanisms and models on aggregating web services, however forming optimal and stable coalitions to maximize individual and community efficiency and income has got less attention. In this paper, we propose an efficient coalition formation mechanism using cooperative game-theoretic methods. We propose a mechanism for membership requests and selection of web services in the scenarios in which there are already established communities. Moreover, we analyze the scenarios in which we do not have pre-defined communities and we 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 simulation results show that our algorithms provide web services and community owners with applicable and near-optimal decision making mechanisms.**

***Keywords*-Web services; Cooperative game thoery; Community of services;**

I. Introduction

Over the past years, web applications have become part of critical buisiness services. The increasing reliance on web-based applications has significant implications for service providers and its users. Web services are methods of communication between electronic devices over the internet. Web service as a highly available software function acessable at a network address over the web or the cloud usually provides a set of *stateless* functions.

TODO on functional and non-functional requirements specially for critial applications

The need for highly available and highly responsive services online has called for collaborative communities of these loosely-coupled business services. The idea of web service communities and the way they are engineered to collaborate was proposed in [1], [2], [3] where web services with similar functionality are grouped together and users send their requests to communities assuring higher service availability and performance than individual web services. These communities are highly available also more resilience to failure since there is no single point of failure in their architecture.

Most of the recent work on communities of services are either user-centric and focus user satisfaction [4] or there are many works in distributed, grid, cluster and cloud computing which are system-centric and focus on system throughput and performance and utilization of the system. However in real world environments both users and services are selfish and self-interested agents, aiming to maximize their own profit. In real world environments, both these parties will participate and collaborate as long as its economically efficient for them.

Recently, [5], [6] have proposed mechanisms in which try to maximize gain for both parties. A two-player non-cooperative game theory method between web services and community manager was proposed in [6]. The strategies for a web service when facing a new communities, is joining or staying in their community and the set of strategies for communities are allowing to resusing the the web service proposal. Both parties based on their capacity, market share and reputation will have different set of utilities over the strategy profiles of the game. The problem with this method is the game is between community master and the new web service, and all other web services already residing within the community are ignored. Those web services can lose utility and gain having to collabrate with new webserices, they may deviate and join other comuunities if they are unsatisfied. In [5] a 3-Way satisfaction selection approach has been proposed. In thier approach, they proposed a web service selection process for community masters catering for the effciency of 3 parties of users, web services and communities. In this work, they show how the gain of these parties are coupled together and they suggest a linear optimization process for optimizing the needs of the 3 parties. However, the optimization problem in this solution tends to optimize some parameters considering all web serivces even if they are weak. There are no clear thresholds for accepting or rejecting new web services. The optimization result for example could suggest web services already residing within the community to increase or decrease their capacity to adjust with weakness of other parties in the system. Hoever, a strong performing web service could deviate anytime it finds itself unsatisfied within the community instead of adjusting its service parameters.

In this paper, we proposed a cooperative game theory based model for aggregation of web services within communities. The solution concepts of cooperative game theory seeks to find most efficient ways of forming coalitions (teams) of agents, maximizing their gain and payoff, and also distributing gain in a fair way to keep coalitions stable by having all agents satisfied. We first propose a representation function for communities of web services based on their qualities and capacities. Using this function we can evaluate the $worth$ of each coalition of web services. Our community managers facing new membership requests, analyses whether the new coalition having the new set of web services will keep the community stable for all web services already residing in the community or not. The community manager will reject the membership requests if it finds out new coalition can be unstable since there can be any subset of web services gaining significantly more by deviating, colluding with new web services and joining other communities. Also web services facing communities, will try to join communities which maximizes their gain, if they find out they can deviate and agin more, they will form new coalitions with other communities. The cooperative game theory solution methods are combinatorial in nature, making them impractical in real world applications; however, using the concepts of group stability we have offered approximation algorithms running in polynomial time providing web services and community owners with applicable and near-optimal decision making mechanisms.

The rest of paper is organized as follows. Section II describes architecture and parameters and some basic cooperative game theory concepts, section III presents our solution model in three different scenarios. Section IV presents our experiments and results and finally section V concludes the paper and identified future work.

II. Preliminaries

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

A. The Architecture

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

1) Web services are rational entities 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. Web services can join communities to have better efficiency collabrating with others, have access to higher market share and opportunity to receive a bigger task pool from end users. The way the web services reside inside communities and how communities of web services are engineered is described thoroughly in [1].

2) Master Web Services are responsible for managing and are representative of the communities of web services.

Table I
List of web service QoS parameters.

Notation	Definition
$ResponseTime$	The average response time of services
$Availability$	Probability of available during a time frame
$Reliability$	P of successfully handling tasks during a timeframe
$Successability$	Probability of handling tasks successfully
$Throughput$	Average rate of handling tasks
$Latency$	The average latency of services
$Capacity$	Amount of resources available
$Cost$	Mean service fee for cooperating

Communities receieve tasks from users and they aim to have a healthy set of web services to perform the tasks. They seek to maximise user satisfaction by having tasks accomplished with good rate and quality. Higher user satisfaction will bring more user tasks and increase the market share of the community.

3) Users generate tasks and try to find service providers online to perform the tasks. User satisfaction is abstracted as function of quantity and quality of tasks accomplished by our service providers.

B. Web Sercive Parameters

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

We adopted a real world dataset [7] which has normalized and aggrigated each of these metrics to a real value between 0 and 1. Since tasks are not shared between web services and are distributed among all web serivices inside a community, one can expect tasks within a community to be performed by average quality of web service residing in that community. We use this average expected quality output, later in evaluating community $worth$ or $payoff$ function.

C. Cooperative Game Concepts

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

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

The second condition is $feasibility$, according to equation below the payoff for each agent cannot be more than coalitions total gain. A payoff vector is also $efficient$ if all the payoff obtained by a coalition is distributed amongst coalition members: $\sum_{i \in S} x_i = v(S)$. This definition of characteristic function works in $transferableutility$ (TU) settings, where utility (payoff) is transferable from one player to other, in other words players have common currency and a unit of income is worth same for all players [8].

When dealing with cooperative games, two issues need to get addressed: 1. What coalition to form (Canonical games)? 2. How to reward each member when a task is completed (Coalition-formation games)?

Definition 1 (Shapley value) Given a cooperative game (N, v), the Shapley value of player i is given by[9]:

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

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

Definition 2 (Core) A payoff vector x denoted by C(N,v) 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 if is not empty is basically a set of payoff vectors (x) where no subset of players S' could gain more by deviating and making their own coalition $\sum_{i \in S'} x_i \geq v(S')$. The sum of payoffs to the agents in any sub-coalition S is at least as large as the amount that these agents could earn by forming a coalition on their own. In a sense it is analogue to Nash equilibrium, except that it about deviations by groups of agents. The core is the strongest and most popular solution concept in cooperative game theory, however its computationally very heavy and is not computationally tractable as number of players increase. The core of some real-world problem games may be empty, which means having characteristic function v there might be no possible distribution of payoff assuring stability of subgroups.

Definition 3 (Convex cooperative games) A game 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 [10] 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 core set is empty, it means no coalition of players can gain anything by deviating, it is the strongest stable condition. An outcome would be unstable if a coalition can benefit even by a small amount from deviating. This is a strong requirement. In some situations, deviations can be costly or agents may have some 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 exceeded a particular cost for example the costs of deviation. $\epsilon - Core$ relaxes the notion of the core, and only require that no coalition can benefit significantly or within a constant amount(ϵ) by deviating.

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

Coalition Structure Formation: Coalition structure formation is the problem of finding best partition of agents into teams, focusing more on optimizing the coalition's "owner" payoff. In these settings performance of individual agent is less important than the *social welfare* of the system, which is the sum of the values of all teams. Having characteristic function game G, a coalition structure (CS) is *socially optimal* is CS belongs to set $argmax_{CS \in CS_N} v(CS)$ where v(CS) is sum of the total valuation function of all coalitions inside CS ($v(CS) = \sum_{C \in CS_N} v(CS)$). The outcome of a characteristic function game in coalition structure settings, consists of two parts; first a disjoint partition of players (agents) into coalitions, called a *coalition structure* (CS) and second a *payoff vector* as mentioned in cooperative game solution concepts, which distributes the value of each coalition among its members.

III. Problem Formulation and Modeling

In this section, we present the architecture of different web service community models and formulate how both web services and community masters as rational agents would maximise their payoff.

A. Web Services and A Grand Community

In this scenario, there are is a community controller by its master web service. The community master has some request rate (R_{master}) from users. Web services need to join a community to be able to get tasks from a master web service. Each web service comes with different quality of service parameters and a throughput (T_{ws}). Throughput is the average rate of tasks a web service can perform. Therefore, the master web service is providing tasks with rate of R_{ws} and web services perform tasks with some QoS output metrics and a rate or T_{ws}. In this setting, we define the value of the coalition as follows:

$$output(C) = \sum_{ws} (T_{ws} \times QoS_{ws})$$

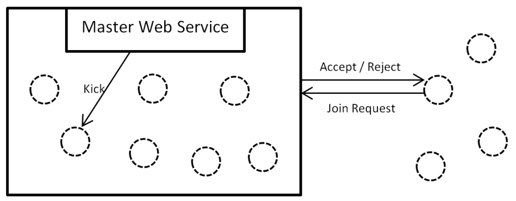

Figure 1. Web Services and A Grand Community

$$v(C) = \begin{cases} output(C) & \text{if } \sum_{ws} T_{ws} \leq R_{master} \\ output(C) \times \frac{R_{master}}{\sum_{ws} T_{ws}} & \text{if } \sum_{ws} T_{ws} > R_{master} \end{cases}$$

The output of a coalition of web services is the rate and quality of task they can perform. If the rate is more than master's input task rate, it means the web services inside the community are capable of doing more tasks than exists. Considering this the valuation function will balance the output performance and will be the exact amount and quality of work they can perform within the particular community.

In the first scenario, we only consider one grand coalition and analyse the system from 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 income or tasks among its community members.

The membership decision is made based on throughput and QoS of the requested web service. The goal is to have quality web services in community so the community stays stable and no other web services would have incentives to deviate and join "other" coalitions. Therefore, we it check the core membership of the coalition of all current community members and the new web service. Our algorithm uses the Shapley value distribution method as described in equation 1 to distribute gain of $v(C)$ among all members and then checks if Shapley value payoff vector for this community having characteristic function $v(C)$ is Core stable or not. In Shapley value payoff vector, the payoff for each web service i, is calculated based on their contribution $v(C \cup i) - v(C)$ and then averaging over the possible different permutations in which the coalition can be formed, this is why Shapley value distribution is fair. It is proven in convex games the Shapley value lies in the core [11], [8]. Therefore if the Core is non-empty, the Shapley payoff vector is a member of Core.

Proposition: The folowwing statement is equivalent with convex condition of equation 3.

For $\forall S \subseteq T \subseteq N \setminus \{i\}, \forall i \in N$ we have:

$$v(S \cup \{i\}) - v(S) \leq v(T \cup \{i\}) - v(T) \quad (5)$$

Proof:

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

We know $T \subseteq N - \{i\}$, therefore:

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

and we have $S \subseteq T$, therefor:

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

With a variable chnage of S we have:

$$v(S \cup \{i\}) + v(T) \leq v(T \cup \{i\}) + v(S)$$
$$\rightarrow v(S \cup \{i\}) + v(T) \leq v((S \cup \{i\}) \cup T) + v((S \cup \{i\}) \cap T)$$
$$\rightarrow v(S') + V(T) \leq v(S') + v(S' \cap T)$$

This means in order to keep the characteristic function convex, new agents should have more marginal contribution as coalition size grows.

Our algorithm works as follows. We have an established community with a master and some (slave) web services already residing in community. Now we have a request from a web service to join our community. The ideal solution in our method would be analyzing the core or $\epsilon - core$ stability of group having this new member. However the normal core membership algorithms are intractable. We exploit the equation 5 to check convexity of our characteristic function game having the new member. In that equation we set T to be our community members (C) before having the new web service. We set i the new web service, and then verify the equation for S, setting $S = T/W$ where W is all 1-member subset of set C. We can relax the equation a bit by adding a constant epsilon to left side of equation. We call this method *Depth-1 Convex-Checker* algorithm. If the equation is true for all W we let the member join our community, since he will contribute good enough to make our new community stable. The run time complexity of this algorithm is O(n) which is huge reduction from $O(2^n)$ checking all possible subsets of N. In our second method, we use the same algorithm however this time we set W to be all possible subsets of size two and one of C. We call this method *Depth-2 Convex-Checker* and its run time complexity is $O(n^2)$. It is possible to develop an anytime algorithm by continuing verifying this condition against all subsets of size 3 to N, and stop whenever we are interrupted.

B. *Web Services and Many Communities*

In this scenario we consider multiple communities owned by multiple master web services which provide independent request pools. Like first scenario master web services form coalitions with web services. We use coalition structure formation methods to partition web services into non-empty

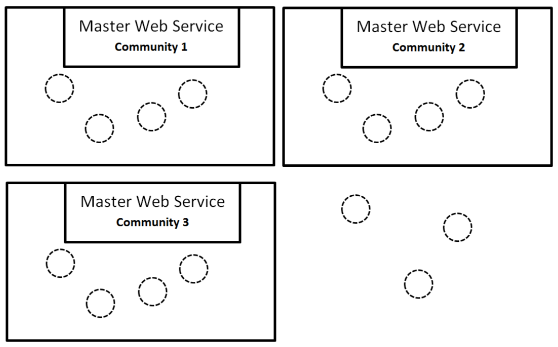

Figure 2. Web Services and Many Communities

disjoint coalition structures. As mentioned in section II-C these algorithms [11], [12] try to solve two fundamental problems of what coalitions to form, and how will the members of these coalitions distribute their total gain to make these coalition structures stable.

The valuation function $v(C)$ for this scenario is same as *one grand community* scenario, however in order to prevent master web services joining same community we set $v(C) = 0$, where C has two or more master web services as members.

One of the properties of coalition structure formation algorithms in our scenario is that they partition web services with low throughput rate usually join coalitions with less request rate. Since the characteristic function $v(C)$ and the fair payoff vector of Shapley value 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 valuation function $v(C)$ web services with high throughput will not contribute well to communities with low amount of user requests. The strong web services are much more likely to deviate weak coalitions, joining a stronger coalition, making the initial coalition unstable.

In coalition-formation games, formation of the coalitions is the most important aspect of in the game. These solutions focus on maximizing social welfare. For any coalition structure π, let $v_{cs}(\pi)$ denote the total worth $\sum_{C\in\pi} v(C)$ which we call it social welfare. The problems in this area deal with finding the maximum value for social welfare over all the possible coalition structures π. There are $centralized$ algorithms for this end, however these approaches are generally NP-complete. The reason is that number of all possible partitions of a set N grows exponentially with the number of players in n, and the centralized algorithms need to iterate through all these partitions. In our application, we prefer not having a centralized decision maker dictating all web services their decisions. Becuase of this plus the complexity issues we would prefer a distributed algorithm where each community master can be a decision maker. The aim is to find low complexity and distributed algorithms for forming coalitions[13], [14], [15]. The distributed merge-and-split algorithm in [13] suits our application very well. It keeps splitting and merging coalitions where it is preferred by all players inside those coalitions. This merge-and-split algorithm is designed to be adoptable for different problems. It requires two functions to be specified, first a preference relation or comparison relation for comparing two collections of different coalition partitions of same players and second function is to assess the stability condition of a partition. Having two partition sets of a subset of our players, namely $P = P_1, ..., P_k$ and $Q = Q_1, ...N_l$ one example would be to use the social welfare comarison $\sum_{i=1}^{k} v(P_i) > \sum_{i=1}^{k} v(N_l)$. For our scenario we used $Paretoorder$ comparison which is an individual-value order appropriate for our self-interest agents (web services). In Pareto order, an allocation or partition P is preferred over another Q if at least one player gets more payoff in new allocation without other players getting hurt, or other players at least same payoff. ($p_i \geq q_i$ with at least one element $p_i > q_i$). As new web services are discovered and get ready to join communities, this algorithm keeps merging and splitting partitions based on the preference or comparison function provides and will terminate[16] as the partitions satisfy the stability condition or no further merging and splitting is found satisfying the comparison function. More details of this algorithm and analyses of generic solutions on coalition formation games are described in [13].

C. Web Services collaborating independently

In this scenario, we do not consider pre-defined communities and master web services. Here web services are interested in forming coalitions to perform better and share benefits. We do not consider concept of master web services providing requests for others, each web service comes with its own request load which in this scenario we call it *market share*. Therefore coalitions will form is web services working together can perform and gain more than working alone individually. The worth or value of coalitions of web services is different in this setting, average the QoS metrics of web service, multiply it by a weight where the importance or weigth of each metric is dependent on type of users and requests web services are dealing with. For each web service we multiply this by web service's market share (MS_{ws}) and the summation would be the characteristic or valuation of independent web service coalitions.

$$v(C) = \sum_{m=metrics} \Big(\sum_{ws \in C} QoS_{ws}^{m} \times MS_{ws} \Big) \times w_m \quad (6)$$

w_m is the weight of important for a QoS metric, it gives the community the flexibility to emphasize or neglect a specific QoS parameter. Note that $\sum_m w_m = 1$. In this scenario we have no master web services, we assume they

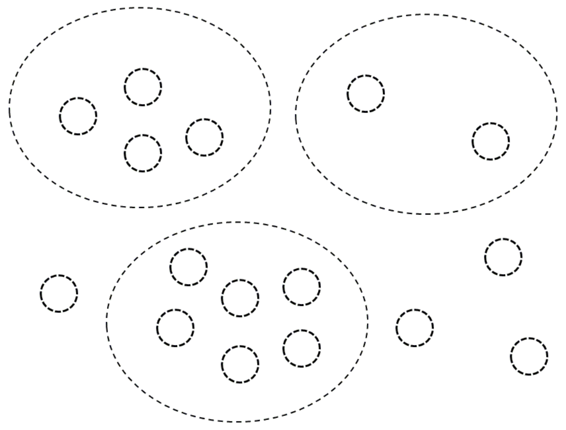

Figure 3. Web Services collaborating independently

either dynamically vote for one, or collaborate and agree on decisions and distribute task in round robin or fair fashion. We use the same merge-and-split algorithm like “Web Services and Many Communities” scenario, to find stable coalitions of web serivces.

IV. Experimental Results

In this section, we discuss the experiments we performed for our scenarios to validate 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 [7]. We programmed the simulations using Java and executed the simulations on a Intel Xeon X3450 machine with 4GBs of memory. For other parameters we used normal distribution with parameters that make most sense in real world examples.

We first analyze the most important performance result our communities should perform, quality and quantity of tasks successfully performed by our communities. This is the most important criteria for user satisfaction. We initiate the community with few web services, then let rejecting and accepting of random web services go for a while, after that we start task distribution for the communities and let them allocate tasks for web services. Then we measure the average output performance of tasks in communities following different methods.

Figure 4 depicts the results of simulation of optimal $\epsilon - Core$ with 10% of total community value deviation allowed (one grand community with many web services), *Depth-1 Convex-Checker*, *Depth-2 Convex-Checker*, 3-Way Satisfactory[5] and *2 Player Non-Cooperative* methods. The the *optimal* ϵ*-core* method we capped the coalition size to 23 web services, since anything more than that would require analysis with time complexity of $O(2^22)$ which would made that impractical to run in our simulations. In other methods

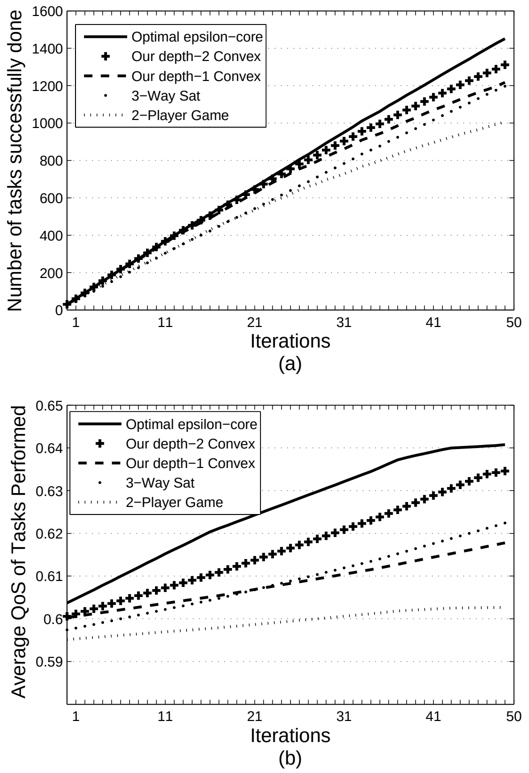

Figure 4. Part (a): Cumulative number of tasks succesfully done. Part (b): Average QoS of tasks performed.

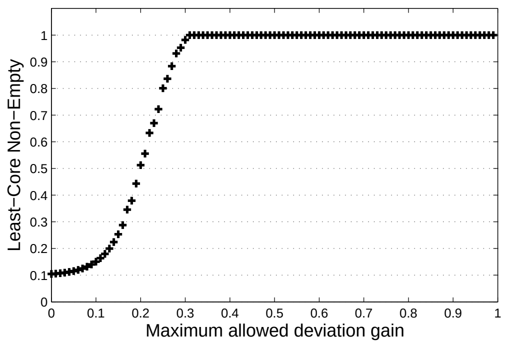

Figure 5. Smallest value of ϵ such that the $\epsilon - core$ is non-empty

there were no cap on size of the community and we had communities of size 60 web services at some points. The results show depth-2 and 1 convex checker methods are performing well compared to other methods and all not far from optimal.

As mentioned in preliminaries section II, core assumes no coalition of players can gain anything by deviating which is a fairly strong requirement, that is why the notion of $\epsilon - Core$ was introduced. Least-Core $e(G)$ of a game G, is the minimum amount of ϵ in which the core is not empty. We evaluated the least-core value using our valuation function

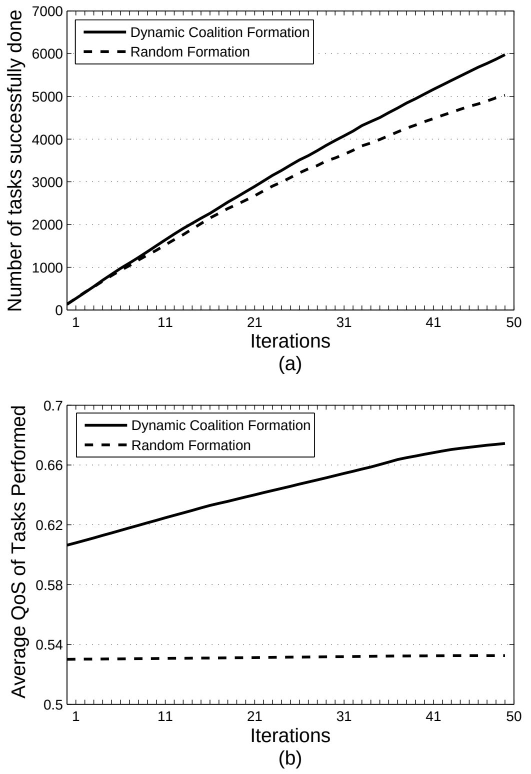

Figure 6. Part (a): Cumulative number of tasks succesfully done. Part (b): Average QoS of tasks performed.

and our set of web services. We pick random number of web services from our data set and form random coalitions consisting of 3 to 25 web services. We choose 25 as maximum number of members in our grand coalition since it is computationally very complex for bigger coalitions to compute least-core. We set ϵ to a ratio of total payoff of coalition v(C) from 0 to 1. The results in figure 5 show, almost in 10% of random web service coalitions have non-empty core solution and least-core solution is always non-empty letting agents gain only 30% more by deviating.

In figure 6 we compare our *Web Services and many Communities* scenario with a method which ignores QoS parameters and forms coalitions and allow web services to join only if they have enough requests for them in other words web services can join a community when request rate is less that throughput of all web services inside community ignoring QoS parameters. We name this method *Random Formation* and use it as our benchmark for our QoS aware coalition formation process. As results illustrate clearly, our QoS aware method, forms better coalitions of web services improving performance and satisfaction for both web service and coalitions.

V. Conclusion

In this paper, we proposed a cooperative game theory based model for aggregation of web services within communities. The goal of our services is to maximize efficiency of performing tasks by collaborating and forming stable communities. 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 goal is to increase revenue by improving user satisfaction, which comes from the ability to perform tasks with higher quality and quantity. 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 work on the convexity condition for the characteristic function on web service applications. A future research direction includes working on compact or succinct representation of our service model to reduce the input complexity of cooperative game stability and fairness and membership checking solutions. From web service aspect 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.

References

[1] 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.

[2] 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.

[3] 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.

[4] B. N. Chun and D. E. Culler, “User-centric performance analysis of market-based cluster batch schedulers,” 2002.

[5] 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.

[6] 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.

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

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

[9] 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.

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

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

[12] 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.

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

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

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

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