Decentralizing the Cloud: How Can Small Data Centers Cooperate?

Hao Zhuang, Rameez Rahman, and Karl Aberer
LSIR, École Polytechnique Fédérale de Lausanne (EPFL)
hao.zhuang@epfl.ch, rameez.rahman@epfl.ch, karl.aberer@epfl.ch

***Abstract*—Cloud computing has become pervasive due to attractive features such as on-demand resource provisioning and elasticity. Most cloud providers are centralized entities that employ massive data centers. However, in recent times, due to increasing concerns about privacy and data control, many small data centers (SDCs) established by different providers are emerging in an attempt to meet demand locally. However, SDCs can suffer from resource in-elasticity due to their relatively scarce resources, resulting in a loss of performance and revenue. In this paper we propose a decentralized cloud model in which a group of SDCs can cooperate with each other to improve performance. Moreover, we design a general strategy function for the SDCs to evaluate the performance of cooperation based on different dimensions of resource sharing. Through extensive simulations using a realistic data center model, we show that the strategies based on *reciprocity* are more effective than other involved strategies, e.g., those using prediction on historical data. Our results show that the reciprocity-based strategy can thrive in a heterogeneous environment with competing strategies.**

I. Introduction

Cloud computing has become hugely popular in the IT marketplace because of its on-demand resource provisioning, high availability and elasticity. These features allow cloud end-users to access resources in a pay-as-you-go manner and to meet varying demands without upfront resource commitments [1]. Currently, many small data centers (SDCs) are springing up in order to avoid severe issues (e.g., legal, privacy, and data control [2]) that can arise with the adoption of massive centralized data centers. According to recent news reports [3], many European countries are now concerned about their data leaving European borders and are also stressing on the need for *local* small data centers to serve local needs. For instance, in a country like Switzerland with various autonomous cantons, one could imagine the emergence of multiple small data centers in various government institutions to obviate the reliance on big enterprise data centers and to maintain control over local data.

However, these small data centers, due to their size, are likely to suffer from resource under-provisioning thus failing to meet peak demand. We term this the *Resource Provisioning Dilemma* faced by SDCs in that they have to make the tradeoff between request loss and the cost of over-provisioning. One way out of this dilemma is for such small data centers to cooperate with each other to help meet each others' user demand, thereby increasing their resources without having to invest in more. Such cooperation is analogous to *Business Clusters* described in mainstream economics which emerge due to, among other factors, shared interests and geographical proximity [4]. We note here that since one of the key benefits of SDCs is the avoidance of legal and privacy issues, it is apparent that SDCs much like *Business Clusters* would prefer to cooperate with those SDCs which lie under one legislative control, thereby alleviating any of their own or their customers' privacy concerns. An alternative could be a big data center under one legislative control, however in practice due to bureaucratic obstacles and lack of centralized coordination, most local enterprises such as universities, research centers, hospitals, and govt. offices, etc., often resort to employing their own SDCs. Hurdles in centralizing data control in general due to organizational mismatches and lack of incentives have been formalized in [5].

Some previous works [6], [7] have focused on platform design for cooperation between cloud providers. Other works have focused on prices and revenue maximization for incentivizing data centers to cooperate [8]. However, in a dynamic setting of data centers with varying workload patterns and changing sociopolitical and legal realities, prices and revenue maximization are not the sole (or at times even relevant) criterion that can determine cooperation. Factors such as *location proximity*, *workload similarity*, *privacy concerns* etc., can be equally, if not more, important considerations for resource exchange and allocation between small data centers.

In this paper, we propose a decentralized cloud with a swarm of networked SDCs which cooperate with each other under varying workloads, and which employ various strategies in order to cooperate with others. Our analysis aims to answer the following questions: i) what are the incentives of SDCs for cooperation within the decentralized cloud; ii) how is the performance of cooperation affected due to different strategies adopted by SDCs; iii) what type of strategy can thrive in a heterogeneous environment. Specifically, the main contributions of this paper are summarized as follows:

(1) We propose a model for decentralized cloud with a swarm of networked SDCs. Three workload arrival models are introduced to simulate different levels of workload burstiness in the SDCs.

(2) A general strategy function that can be employed by SDCs is proposed to evaluate the performance of cooperation between the SDCs. Moreover, we design four specific strategy functions based on different dimensions: *capacity, history, prediction* and *reciprocity*, to model conditions under which SDCs can choose to cooperate with each other.

(3) Through extensive simulations of realistic models of data centers, we analyze the performance of various strategies from the perspective of both the decentralized cloud as a whole, and individual SDCs. We discover that the strategy with the best performance is the simplest strategy which is

based on reciprocity. Furthermore, we design a new adaptive approach for SDCs to learn how to select a strategy effectively in a heterogeneous environment. The results reveal that most of the SDCs will eventually converge on the same strategy in order to achieve the state of stable mutual cooperation.

II. Related Work

Work focusing on decentralizing the cloud includes [6], which proposed decentralized cloud that uses volunteer edge resources. Wang *et al.* [9] study a cloud platform built on customer-provided resources called SpotCloud, which allows customers to sell their idle resources to offer cloud services collaboratively. Other approaches [7] design and implement a decentralized cloud computing platform in an Infrastructure-as-a-Service level, based on P2P technologies, while our work focuses on studying the incentives and strategies of resource sharing among the SDCs and on how the SDCs can cooperate in a decentralized manner.

Our work is not alone in exploring the resource sharing models in the multi-cloud environment. However, most of the previous studies are based on the federated cloud environment which often relies on centralized coordination with limited number of cloud providers [8], [10]. Samaan [8] presents an economical resource sharing model, which is based on Markovian model, in a federation of selfish cloud providers (CPs). However, her model relies on a centralized federation broker which performs the VM allocations among the cloud providers. The policies in [10] focus on helping providers to make the decisions on executing a job locally or remotely to improve providers' profit. On the other hand, [11] propose resource sharing models based on brokerage services, which assume a centralized broker to aggregate the resource demands and to bridge the gap between the cloud providers and users. Our work is distinct from these in that we do not assume a federated cloud of selfish cloud providers with full rationality and where resource sharing is only based on monetary payment.

The decentralized cloud model introduced in our work combines the advantages of both cloud federation model [12] and peer-to-peer (P2P) model. It is similar to P2P in that it is decentralized and self-organizing for the aggregation of resources to meet user demands. Instead of using monetary payment, our model uses reciprocity-based scheme as an incentive for sharing resources, which is similar to the Tit-for-Tat strategy in file swarming systems [13]. The strategies designed in our paper are inspired by Axelrod's work on the conditions for the emergence of cooperation [14]. Nevertheless, compared to normal P2P systems, the number of SDCs is much smaller and the computing environment of our model is not so highly dynamic as a P2P environment. Thus, in contrast to P2P, we do not face the problem of churn, and the participation of SDCs is more stable. Also, other factors such as legal concerns for choice of partners can be of importance, which is not the case in P2P. We can design a general strategy function, which incorporates various factors of resource sharing, to evaluate the performance of cooperation. Finally, we are not aware of other works that study the effect of different cooperation strategies on service satisfaction in the decentralized cloud.

III. Model Description

A decentralized cloud computing system consists of a swarm of networked small datacenters (SDCs). Each of the SDCs has different capacities, various workload arrival patterns and diverse strategies for resource provisioning. In this section, we will explain the models in our paper.

A. Small Data Center

We assume that an SDC has a certain number (hundreds) of networked servers which can host multiple virtual machines (VMs). Each VM requires a set of resources, including CPU, memory and storage to serve the workloads which are submitted by cloud end-users. A *workload* is defined as a resource request from cloud end-users, with variable execution time. The different SDCs have different levels of request burstiness, which can be modeled with various workload arrival models. We assume a discrete time horizon, $t \in \{0, 1, \cdots, T\}$. After receiving workload requests at time t, an SDC will find available resources among all its physical servers, which will be returned as VMs, to serve user requests. If all the servers within a single SDC are busy, the SDC will reject the resource requests, which results in revenue loss and user dissatisfaction. In our cloud model, we assume that the performance, in terms of satisfied requests, of each SDC depends on its workload arrival and free capacity. From the perspective of resource elasticity, the *request loss* of an SDC i is defined as the accumulated amount of under-provisioned resources, which is formulated as:

$$loss_i = \sum_{t=0}^{T} max(d_i(t) - c_i(t), 0) \tag{1}$$

where $d_i(t)$ is the number of resource requests and $c_i(t)$ is the free capacity of the SDC i at time t. $d_i(t)$ and $c_i(t)$ vary with time t and the SDC i is under-provisioned when $d_i(t) > c_i(t)$. To minimize the loss of request, the SDC is forced to optimize its capacity and demand planning (e.g., buying more resources or shifting the demand). However, we will later discuss how SDCs can alternatively reduce their request loss through cooperation within the decentralized cloud. It is noted that we do not consider the impact of quality of service (QoS) on the request loss. We assume that VM techniques provide good performance isolation so that the QoS remains unaffected with the fluctuation of requests.

B. Workload Arrival Models

In this section, we will discuss the workload arrival models based on three types of stochastic arrival processes, namely Poisson process, Markov-modulated Poisson process and heavy-tailed process. These models have been extensively used for modeling real world traces of job arrival in cloud datacenters [15]-[16]. With three different arrival models, we can simulate the SDCs with different levels of workload burstiness and we assume the arrival of workloads at different SDCs is independent from each other, i.e., they are independently sampled. We believe that this is a fair assumption since individual SDCs are independent of each other and can have different sets of customers with different demand patterns.

1) Poisson Arrival: The Poisson arrival process [15], [16] is a counting process which counts the number of events in a given time interval. In our cloud scenario, an event is the request of a VM from the SDC. Thus, the workload arrival process of an SDC, $\{D(t), t \geq 0\}$, is a Poisson process with the mean rate λ if:

$$P[D(t+\tau) - D(t) = k] = \frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!}, k = 0, 1, ...,$$

where k is the number of requests in the time interval $(t, t+\tau]$. The Poisson arrival has a low level of burstiness due to the independent increments property, i.e., the number of requests arriving into the system after time t is independent of the number of requests arriving into the system before time t.

2) Markov-modulated Poisson Process: A Markov-modulated Poisson Process (MMPP) [15]-[16] is a doubly Poisson process whose rate varies according to a Markov process. It is particularly useful in modeling the process with the time-varying arrival rate. We consider a Markov process with state space $\{1, ..., r\}$ where each of the r states corresponds to an arrival rate λ_i, i.e., when the Markov process is in state i, the rate of MMPP is λ_i. In our model, we consider a simple MMPP with two states, namely high (λ_h) and low (λ_l). The transition probabilities between high and low state are denoted as p_h and p_l. Thus, the expected rate of $D(t)$ is $\lambda = \frac{p_h}{p_l+p_h}\lambda_h + \frac{p_l}{p_l+p_h}\lambda_l$. We can control the level of burstiness through adjusting the transition probabilities between the high and low rates.

3) Heavy-tailed Arrival: We consider the case of heavy-tailed arrival [15], [16] in which we model the burstiness of requests as Bounded Pareto distribution. A Bounded Pareto Distribution (BP(L, H, α)) is characterized by 3 parameters L, H and α, where L and H are the low and the upper bound for the number of service requests respectively. The probability density function is defined as:

$$f(x|L, H, \alpha) = \frac{\alpha L^\alpha}{1 - (\frac{L}{H})^\alpha} x^{-(\alpha+1)}, L < x < H$$

where $\alpha > 0$ is a shape parameter which is inversely proportional to the variance of the distribution. In our cloud scenario, H, L can simulate the high peak and low peak of the service requests. The expected request rate is $\lambda = E(X) = \frac{L^\alpha}{1-(L/H)^\alpha} * \frac{\alpha}{\alpha-1} * (1/L^{\alpha-1} - 1/H^{\alpha-1}), \alpha \neq 1$.

***Discussion**:* To compare the three different types of workload arrivals, we set the parameters to have the same mean workload arrival rate. As is shown in Figure 1(a), the mean arrival rate of all three workload arrivals is around 100, but they evidence different levels of burstiness. The Poisson arrival with $\lambda = 100$ is the most stable which fluctuates around the

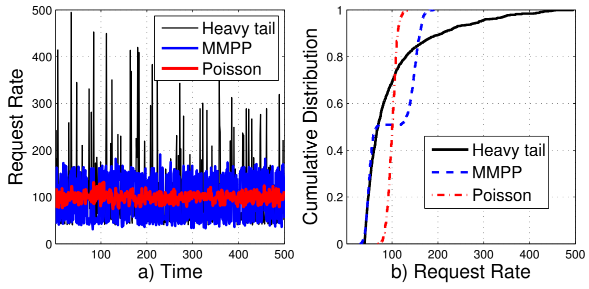

Fig. 1: Characteristics of the three workload arrival models

average rate while the Heavy-tailed workload varies drastically between the low rate ($L = 40$) and the high rate ($H = 500$). Figure 1(b) shows the cumulative distribution of arrival rate. In this Figure, we can see that MMPP has about 50% of arrival rates around high rate ($\lambda_h = 150$) and the remaining half around low rate ($\lambda_l = 50$) since the transition probabilities are the same ($p_l = p_h = 0.5$). We set $\alpha = 1.2$ which results in about 20% of request rates greater than the average.

Queueing theory has been traditionally used in request arrival and service problems. However, in our three workload models, the service times do not follow the exponential distribution. Thus, we should consider MMPP and Heavy-tailed as two renewal processes while Poisson arrival process in our case is a special case of M/G/n queue. Although some limited analytical derivation for such queue models has been proposed in the literature, their solutions are often mathematically challenging [17]. Taking inspiration from [18], we use a simulation based approach (as opposed to an analytical approach) to evaluate the impact of different workload arrivals.

C. Decentralized Cloud

We consider a decentralized cloud with n networked SDCs. By participating in the decentralized cloud, SDCs can work collaboratively and support each other, i.e., share their unused capacities during low-demand periods or borrow spare capacities during peaks. In other words, all the SDCs in the decentralized cloud form a resource sharing network which can be represented as a weighted directed graph $G = \langle V, E \rangle$. Let P be the set of all SDCs as the vertices (i.e. $P = V$) and the edges are represented as the cooperation between the SDCs. For example, an edge $e_{ij} \in E$ connects two SDCs $p_i, p_j \in P$ in the direction $i \rightarrow j$. In addition, the weights w on the edges $e \in E$ are given by a *strategy function* $v_e(\cdot)$ which evaluates the cooperation between two SDCs in an edge. Therefore, the weight of an edge connecting two SDCs indicates how well they cooperate together. We do not assume that all the SDCs share the same strategy function since different SDCs may have different ways to evaluate the cooperation with their partners. Instead, we define the strategy function $v_e(\cdot)$ as a *black-box* function which can evaluate cooperation in terms of different dimensions of resource sharing. For example, the strategy function can take factors such as location proximity, network latency and workload arrival patterns into consideration. We will further discuss the strategy function in the next section.

Furthermore, we define the *partners* $\Gamma(p)$ of an SDC $p \in P = V$ as the set of SDCs who can share the resource with p. In our model, we assume the partnership is symmetric (i.e., if $p_j \in \Gamma(p_i)$, then $p_i \in \Gamma(p_j)$) and non-transitive (i.e., if $p_i \in \Gamma(p_j)$ and $p_j \in \Gamma(p_k)$, it does not mean $p_i \in \Gamma(p_k)$). In other words, an SDC cannot share the resources with a partner of its partners. We also assume that resource sharing among the SDCs in the decentralized cloud is free of charge rather than based on monetary payment scheme [19]. All SDCs maintain their local history of the interactions with their partners and use this information to evaluate the performance of cooperation. Furthermore, each SDC makes its own decision independently based on its strategy function and there is no central authority to monitor their behaviors. In the following, we discuss three modes of decentralized cloud:

1) Non-cooperative: In this extreme case, there is no cooperation among different SDCs, i.e., $E = \{\emptyset\}$. Thus, given any SDC p_i, it will not share its resource with anyone else since its partners set $\Gamma(p_i)$ will be empty. Therefore, the total capacity of p_i at any time t remains the same as its initial capacity at the time 0. Due to the lack of cooperation, SDCs in this mode cannot borrow resources from others. They will suffer losses because of failure to increase their limited capacity through cooperation.

2) Fully-cooperative: In the fully-cooperative decentralized cloud, all the SDCs share their resources and support each other. In this case, given any SDC p_i at any time t, it can share the resources with any other SDCs in the decentralized cloud, i.e., partners set $\Gamma(p_i) = \{p_j \in P | j \neq i\}$. Thus, the total capacity of each SDC can change over the time. However, an SDC in the decentralized cloud can still suffer a loss but only when there are no other SDCs with enough free capacity to support it, i.e., $\forall p_j \in P, \nexists c_j(t) > d_i(t)$, where $j \neq i$.

3) Cooperative: In this mode, each SDC has a limited number of partners (i.e., $\Gamma(p) \neq \{\emptyset\}$) and they can share unused resources and support each other. Moreover, each SDC p_i will evaluate the performance of cooperation with its partners via the strategy function, i.e., $v(e_{ij}), p_j \in \Gamma(p_i)$. Through evaluation, p_i can manage its partners set (e.g., replacing current partners with better ones) in order to achieve different goals, varying from minimizing the request loss to improving the service response time. In this paper, we assume the SDCs initially build the set of partners based on location. We also set the maximum number of partners for each SDC. Thus, the SDC p_i will be under-provisioned only when all the partners in $\Gamma(p_i)$ do not have enough resources, i.e., $\forall p_j \in \Gamma(p_i), \nexists c_j(t) > d_i(t)$.

IV. Evaluating Cooperation with strategy function

In this section, we discuss the strategy function which evaluates the performance of cooperation between two SDCs. It is a black-box function since no SDC has knowledge of the others' strategy functions. Nevertheless, the strategy function accounts for whatever optimization the SDCs can do on dimensions of the resource sharing. We define a finite set X of resource sharing factors. In addition, given the resource sharing network $G = \langle V, E \rangle$ with edge weights w, the strategy function that is associated with an edge e can be formulated as:

$$w = v_e(\mathbf{x}), e \in E, \mathbf{x} \in X \tag{2}$$

where the analytic form of $v_e(\mathbf{x}) : X \to \mathbb{R}$ is unknown and $\mathbf{x}$ represents the different factors associated with SDCs. For example, $\mathbf{x}$ can be a vector of factors such as capacity, workload pattern and network latency. The weight w measured by strategy function indicates the performance of cooperation, which serves two purposes: resource sharing evaluation (RSE) and partnership management (PM). In turn, RSE can be used for two purposes: lending and borrowing. The objective of lending evaluation is to decide if an SDC agrees to lend resources to others while that of borrowing evaluation is to decide from whom an SDC should borrow. For example, if an SDC p_i needs resources, it will send resource requests to all its partners $\Gamma(p_i)$. Each partner in $\Gamma(p_i)$ will evaluate the performance of potential cooperation with the strategy function $v_{e_{ji}}(x)$ and send the results of evaluation to p_i. After receiving all the responses from its partners, p_i will also evaluate the cooperation with $v_{e_{ij}}(x)$ and select the best one to work collaboratively. Second, as for PM, the strategy function provides the results on the performance of cooperation, which helps the SDCs replace less cooperative partners with more cooperative ones. In the following, we will study four types of strategytfunctions:

1) General Strategy: The general strategy is the basic one which takes the SDC's capacity into consideration. This strategy only guarantees that an SDC has enough available capacity to offer its help to others. The SDC with the general strategy is too short-sighted or unconcerned to have second thoughts on the impact of its sharing behavior, which makes it more generous than other strategies. Thus, the strategy takes the user generosity into consideration and has been inspired by [19]. Meanwhile, this strategy is the base one for other strategies since having enough resources is the base for cooperation among the SDCs.

2) History-based: In the history-based strategy, the SDCs need to save the history of past interactions. The reliance on history to make cooperation decisions has been utilized extensively in the P2P literature where they are usually described as indirect reciprocity schemes or reputation systems or community level incentives [20], [21]. We denote the interaction as $r(p_i, p_j, t)$ which indicates the accumulated number of resources that p_i has lent to p_j at time t. Let the global history H be the set of the interactions of all SDCs and H_i is the history saved by SDC p_i. Thus, p_i can assess the behavior of p_j at time t based on

$$H_i^t(p_j) = \{r \in H_i | r = r(p_i, p_j, t) \text{ } or \text{ } r = r(p_j, p_i, t)\} \tag{3}$$

We assume that no SDC can access the global history. Each individual SDC maintains its own history to evaluate its partnership. For example, an SDC may prefer to lend its resource to the SDCs which gave it the maximum support in the past. In addition, given that resource sharing is free (in terms of money), each SDC in the decentralized cloud has an incentive to be selfish, e.g., borrowing more resources than lending. We define the *altruism level* as the ratio of resources borrowed to the amount lent. For the History-based strategy function, an SDC p_i can evaluate its altruism level with other SDC p_j based on its partner set:

$$al_i^j = \frac{r(p_j, p_i, t)}{r(p_i, p_j, t)}, p_j \in \Gamma(p_i), r \in H_i \tag{4}$$

An altruism level of 1.0 means that p_i has borrowed as much resources as it has lent. A selfish SDC, which has borrowed more than what it has lent, has an altruism level greater than 1 while a selfless SDC has an altruism level less than 1. It should be noted that each SDC evaluates its altruism level based on its private history, which is thus a local value rather than a global one. Based on the evaluation results, the SDCs can decide whether to lend the resource to their partners (for RSE), or to replace their partner with a new one (for PM).

3) Prediction-based: Another key determinant to lend resources to others is the arrival of future workload. The strategy function based on prediction will evaluate the impact of resource sharing in a probabilistic way since all three workload arrival models are based on a stochastic process. For this strategy, we have taken inspiration from predictive models for the arrival of workload in cloud which have been studied to allow elastic resource provisioning [1], [11]. At any given time t, after receiving a borrow request, an SDC p_i will evaluate the probability of incoming demand at time $t+1$ greater than its available capacity, i.e., $P(d_i(t+1) > c_i(t+1))$, to decide if it should lend resources. Moreover, we define *risk indicator* as a threshold value to evaluate the risk of prediction. If the probability is greater than risk indicator, the SDC will not lend its resources and reject the requests from its partners. Different SDCs will set different values for risk indicator. For example, if an SDC is risk-averse, it will set indicator as low as possible, in order to minimize the risk of resource under-provisioning.

4) Reciprocity-based : In real cloud scenarios, one SDC usually will likely have a limited number of partners. This can happen due to geographical and legal constraints. The problem is to find out partners with high performance of cooperation. In our decentralized cloud, we initially build a partner set for each SDC merely based on the location. There is the possibility that the workload arrival patterns of two geographically neighboring SDCs are not compatible, e.g., both might reach their demand peaks at the same time, thus leading to poor performance. Since each of the SDCs has limited number of idle resources, it is ideal to develop a cooperative partnership such that the amount of resources lent to their partners can be exchanged for borrowing the same amount of resources. In our decentralized cloud scenario, the SDC with reciprocity-based strategy will replace the partners which reject its requests, with other new partners, which is similar to the Tit-for-Tat strategy in the BitTorrent protocol. It was in their book, *Prisoners Dilemma*, that A. Rapoport and A. Chammah, first introduced the Tit-for-Tat Strategy [22]. This fact often remains unmentioned in the literature. Later on it gained popularity through Axelord's tournament [14]. And finally, it inspired BitTorrent and other P2P systems [13], [23]. We define *tolerance* (τ) as the maximum number of rejections that an SDC can accept from its partners. When the number of rejections of one partner reaches the tolerance threshold, the SDC will replace this partner with a new one. Thus, at any time t, reciprocity-based strategy only depends on the history of last τ moments $H^{t-\tau}$, and thus it is unnecessary to keep all the history.

Name (Abbr.)	x Factors	RSE	PM
General (Gen)	$c_i(t), d_i(t), location$	√	
History (Hist)	$c_i(t), d_i(t), location, H$	√	√
Prediction (Pred)	$c_i(t), d_i(t), location, workload$	√	
Reciprocity (Recip)	$H^{t-\tau}, Location$		√

TABLE I: Strategy function and its usage

All the above four strategies take location into account. In our decentralized cloud, we assume that each SDC prefers to collaborate with the nearest ones, which can improve the service response time. We conjecture that in practice this is how things will turn out since most SDCs would prefer not to cooperate with SDCs in distant geographical locations due to several issues including legal concerns and service response time. Table I summarizes four strategy functions and their main purpose. Furthermore, through combination of 3 RSE strategies and 2 PM strategies, we can have 6 more strategies. In summary, we will discuss the following 9 strategies in terms of three groups (the abbreviation of each strategy is listed in table I): 1) *Strategies without PM*: Gen, Hist, Pred; 2) *Reciprocity-based strategies for PM*: Gen-Recip, Hist-Recip, Pred-Recip; 3) *History-based strategies for PM*: Gen-Hist, Pred-Hist, Hist-Hist.

V. Methodology

A. Simulator

We developed our simulator based on CloudSim [24] which is a simulation toolkit that enables modeling and simulation of cloud computing systems based on discrete events. The simulator consists of three modules, namely *DecentralizedCloud*, *WorkloadManager* and *CloudBroker*. The DecentralizedCloud module implements the networked SDCs, VM allocation and resource sharing protocols among the SDCs. WorkloadManager implements three different workloads and generates workload arrivals for cloud end-users. CloudBroker acts on behalf of cloud end-users, and is responsible for sending resource requests and submitting the workloads to the VMs. The simulator reads a configuration file based on XML format which contains specific values for the various X_i factors of the strategy functions, workload information and network topologies of the SDCs.

B. Experimental Setup

To simulate the decentralized cloud, we set up the basic configurations for the workload arrival models, workload execution time and the SDCs.

Workload Arrival Models: By default, we use the same configuration as discussed in section III-B.

Workload Execution Time: To model the variability in workload execution time, we assume the execution time is distributed as follows: when a workload is generated with probability 0.8, the execution time is uniformly distributed in [0.5, 10] time units; with probability of 0.18, it is uniformly distributed in the interval [10, 50] and finally with probability of 0.02, it is uniformly distributed from 90 to 100 time units. Thus, the average execution time is about 11.5 and the maximum execution time is 100.

Small Data Center: All the SDCs have homogeneous physical server machines (with 6 cores, 24GB memory and 1TB disk space) and each of them can host three VMs (with 2 cores, 8GB memory and 300GB disk space). The total capacity of an SDC is the number of server machines which is uniformly distributed in [370,420]. The distance between two neighboring SDCs is also uniformly distributed from 10 kilometers to 30 kilometers. We note that the network latency between two SDCs depends on the distance since we ignore the cost of data transfer.

C. Metrics

We use the following metrics to evaluate the performance of the decentralized cloud:

Request Loss is the accumulated amount of under-provisioned resources in the decentralized cloud, i.e., $\sum_{i=1}^{n} loss_i$.

Resource Over-provisioning is the accumulated amount of over-provisioned resources in the decentralized cloud, which is computed as the total number of the hours when the VMs in the decentralized cloud are in the idle status.

Service Response Time describes the performance of the decentralized cloud from the perspective of cloud end-users and evaluates how long the cloud-user must wait for a response to a resource request.

VI. Results and Discussion

A. Incentive Analysis

1) Resource provisioning dilemma: We firstly simulate a single SDC to find out the impact of different workloads on resource provisioning. We vary the total capacity of an SDC with three different workloads which are shown in Figure 1. It can be clearly seen in Figure 2 that the increase in the total capacity of an SDC leads to a slow decline in the amount of request losses (Figure 2 (a)) but fast growth in the accumulated amount of idle resources (Figure 2 (b)). From the perspective of resource over-provisioning, we can see all three workload models show similar performance. It is because all three workload arrivals have the same mean arrival rate and therefore the total amount of workloads throughout the simulation is almost the same. The difference between the amount of idle resources under different workload arrivals is caused by the different levels of request burstiness of the three workload arrivals. On the other hand, we can observe in Figure 2 (a) from the perspective of request loss, resource provisioning of an SDC with heavy-tailed arrival also presents a heavy-tailed characteristic. It can be also observed that if an SDC wants to achieve zero-loss of requests, the SDC with heavy-tailed arrival has to increase its capacity to at least 650 while in case of the Poisson arrival, an SDC only needs 450. This implies that in order to accommodate the peak of workload arrival, the SDC with heavy-tailed arrival has to over-provision nearly 0.5 times more than that of the Poisson arrival. Thus, in the real cloud scenarios, the SDCs with large variance in their workload arrivals have to confront the dilemma of resource provisioning. They should consider the tradeoffs between the request loss and the cost of resource over-provisioning. For example, the SDC with heavy-tailed arrival in Figure 2 (a) may only provision for 500 capacity and incur the loss of about 150 resource requests, while saving the cost of maintaining an additional 150 more server machines required to achieve zero loss (at 650 capacity in the Figure 2 (a)).

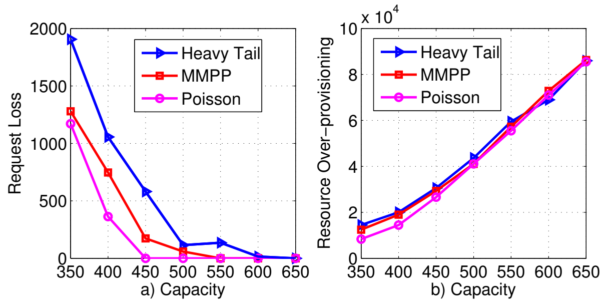

Fig. 2: Resource provisioning with different capacity

Figure 2 can also help us in analyzing the incentives of cooperation in the decentralized cloud. On one hand, there are surplus resources, which is the necessary condition of enabling resource sharing. We can see this in Figure 2 (b), that no matter how the capacity and workload model change, the SDCs on the whole always have a considerable amount of idle resources. On the other hand, there is a great demand of resource sharing to avoid both the loss of resource requests and the monetary cost of buying more resources. Resource sharing in the decentralized cloud provides a fast way to provision additional resources, thereby avoiding the delay in meeting user demand and revenue generation. In fact, in the real world cloud scenario, we believe there are more incentives for cooperation other other revenue growth. For example, finding the resources in specific location improves service response time and/or suitably addresses legal and privacy issues.

2) The Impact of Cooperation: From Figure 2 (a), we can conclude that with capacities in the range [370,420], all the SDCs will suffer request losses if they run individually. In order to avoid these losses, the SDCs can cooperate with each other. To further analyze the impact of cooperation, we do another set of experiments. We simulate a decentralized cloud with 25 SDCs in three different modes to evaluate the impact of cooperation among the SDCs. We randomly assign the workload arrival model and capacity to each SDC. We assume the utilization of each SDC is uniformly distributed in the interval [0.8, 0.9]. With the mean workload arrival rate $\lambda = 100$ and the average workload execution time, we generate the capacities of 25 SDCs which are uniformly distributed from 370 to 420.

Firstly, we simulate the decentralized cloud in a non-cooperative way which means no cooperation exists among the SDCs ($|\Gamma(p)| = \{\emptyset\}$). This will serve as the *worst case scenario*. In Figure 3 we can observe that in the non-cooperative case, the total request loss is as high as about 15,000. Now, we introduce cooperation to the decentralized cloud where all of the SDCs adopt the general strategy to decide whether they share the resource or not. If the decentralized cloud is fully cooperative, i.e., all the SDCs can share resources with each other, the loss will decrease drastically to about 1200, only 8% of the request loss in the non-cooperative mode as can be seen in the Figure. This will serve as the *best case scenario*. We argue that in the real cloud scenario, cooperating SDCs would not be cooperating with a great number of partners due to geographical and legal constraints, among other factors. Therefore, the performance of cooperation between SDCs would be somewhere between the worst-case and best-case scenarios. As a validation of the above point, it can be seen from Figure 3 that with each SDC having 2 resource sharing partners, the request loss of the decentralized cloud will decrease to about 10,000. Moreover, the loss will fall by nearly 40% (of that with 2 partners) if the number of partners increases to 5, i.e., $|\Gamma(p)| = 5$. Both these values are between the worst case and best case scenarios.

Therefore, from the Figure 3, we can find that the cooperation among the SDCs in the decentralized cloud greatly reduces the loss of resource requests. In addition, the more partners an SDC has, the more the resource requests are served in the decentralized cloud. However, there are new problems that emerge. Since different SDCs have different workload arrival patterns, capacities and strategies for resource allocation, the following issues need to be addressed: How can we determine the performance of different strategies? How to form partnerships based on good sharing behaviors and shared

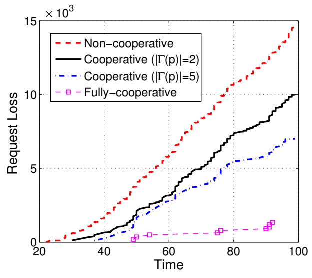

Fig. 3: Impact of cooperation

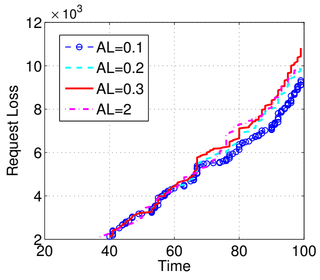

Fig. 4: Varying altruism level

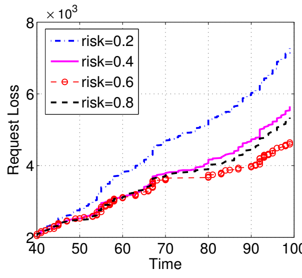

Fig. 5: Varying risk indicator

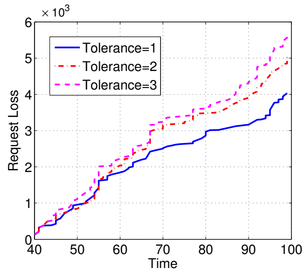

Fig. 6: Varying tolerance

interests and needs? Moreover, does there exist a strategy that outperforms others in a heterogeneous environment, and thus is robust? In the following, we will try to answer these questions with extensive evaluations.

B. The Impact of Strategies

In this section, we will evaluate the three groups of strategies described in Section IV.

1) History-based strategy: An SDC with history-based strategy will evaluate the performance of cooperation based on the past interaction history. Specifically, every SDC will save the borrowing/lending interactions for future decision making. When the SDC (p_i) receives borrow request from the other SDC ($p_j \in \Gamma(p_i)$), it will firstly compute its altruism level (al_i^j) with p_j , which is the ratio of total amount of resources borrowed from p_j to the amount of resources lent to p_j. If $al_i^j > 1$, it means p_i has borrowed more resources than it has lent to p_j. Therefore, the higher the altruism level is, the more selfish p_i is. After computing the al_i^j, it will compare the value with a threshold altruism level. Only if the altruism level is greater than the threshold, p_i will lend its resource to p_j. The effects of varying this threshold value are shown in Figure 4. It is interesting to observe that the smaller the threshold is, the better the performance is. Put another way, this strategy encourages the SDCs to be selfless so that they can suffer less request loss.

2) Prediction-based: Prediction-based strategy enables an SDC to predict future demand. Suppose an SDC p_i receives a borrow request of m resources, it will firstly compute the moving average of workload finishing rate $f_i(t)$ based on history. Then the predicted free capacity of p_i at the time $t+1$ is $c_i(t+1) = c_i(t) + f_i(t) - m$. Then p_i will compute the probability that the next demand $d_i(t+1)$ is greater than the future free capacity, i.e. $P(d_i(t+1) > c_i(t+1))$, based on cumulative distribution function. The probability approximating to 1 means that the risk that p_i cannot serve the demand at time $t+1$ is high. Therefore, if the SDC is risk-averse, it will set a risk indicator as low as possible to minimize the risk. We vary the risk indicator in the interval (0,1) and the results are shown in the Figure 5. We can see that the SDCs with the risk indicator 0.6 have the minimal loss while those with the risk either too small (0.2) or too large (0.8) will suffer more request loss. Thus we discover a *tradeoff* between being risk-averse and risk-seeking. On the one hand, if all the SDCs are risk-averse, they will be reluctant to cooperate so that the prediction-based strategy will reduce to a non-cooperative model. On the other hand, if all of them are risk-seeking, they will accept the borrowing requests even when the risk is high, due to which the performance reduces to that of the general strategy. Therefore, we argue that it is important to have moderate risk estimation to prevent future loss of requests.

3) Reciprocity-based: Reciprocity-based strategy is employed to manage the partner set. At the beginning, we assume each SDC finds its partners based on the location proximity. The maximum number of partners is 5 for each SDC in the decentralized cloud. The SDCs will select new partners based on location if they want to replace the less cooperative partners. However, if no new partners are found, the SDCs will simply remove the less cooperative partners from the set. Meanwhile, each SDC maintains a block list which helps it avoid the selection of removed partners in the future. To improve the forgiveness of the strategy, the block list will be emptied at regular intervals. We vary the tolerance from 1 to 3, respectively. It can be clearly seen in Figure 6 that strategy with the tolerance equaling 1 has the best performance in avoiding request loss. The other two strategies are more forgiving versions in that they do not punish isolated rejections. However, the results show that an excess of forgiveness is costly. The precise level of forgiveness that is optimal depends upon the environment. Though, results show that in our decentralized cloud setting, it is better for the SDCs to set the tolerance as 1 for partnership management.

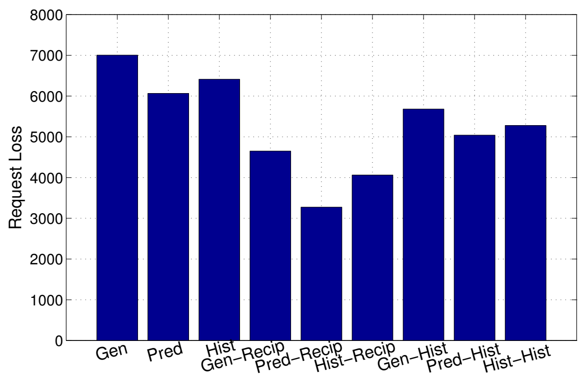

Fig. 7: Performance of combined strategies

4) Strategy combination: Through the evaluation of each strategy, we derive the near-optimal parameter values for each strategy. In the following simulation, we will set the altruism level as 0.1 for history-based strategy, risk indicator as 0.6 for prediction-based strategy and the tolerance as 1 for reciprocity-based strategy. We apply those strategies into RSE and PM with combination and the results are shown the Figure 7.

In this set of experiments, we will vary the strategy of the SDCs while keeping other parameters such as workload and capacity unchanged. We analyze the impact on request loss and service response time. From Figure 7, we can safely reach two conclusions: 1) the strategies with partnership management improve the performance greatly, which means that it is important for the SDCs to find good partners. 2) The various strategies based on reciprocity outperform the others. This property enables the SDCs to manage their partnerships effectively only relying on the last decisions of their partners, rather than the large amount of past interaction histories. Besides that, we also notice that prediction-based strategies for RSE are better than the other two RSE strategies. The reason is that, in decentralized cloud model, we have deterministic workload arrival models. However, in the real cloud scenarios, it is difficult to predict the future demand and therefore the prediction-based strategy may not outperform the other two. Nevertheless, we can still apply the reciprocity-based strategy to achieve better performance.

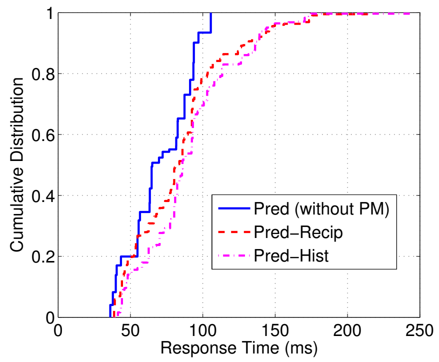

Fig. 8: Impact of Partnership Management (PM) on response time

On the other hand, with introduction of partnership management, the new selected partners may be located far away, which exerts negative influence on the service response time. If we view each discrete time unit as 1 hour, the network latency between two neighboring SDCs is uniformly distributed in [10ms,30ms] which is determined by the distance. In Figure 8, we explore the impact of different PM strategies on the service response time. It shows three strategies with the same Prediction-based strategy for RSE but different strategies for PM. We can observe that compared to the strategy without PM in which only about 8% requests have response time larger than 100ms, the response time of the strategies with PM increases remarkably with about 20%-25% of requests having response time larger than 100ms. In addition, we can observe the heavy-tailed distribution in response time of the strategies with the PM, where a proportion of the response time (about 1%) can be as high as 250ms. Thus, SDCs have to find the tradeoffs between the service response time and the request loss. For example, if an SDC wants to guarantee the service response time for its users, it shall consider the maximum distance that its partners can be located. To put it bluntly, it is not advisable for an SDC in Europe to cooperate with an SDC in Japan (the assumption here being that an SDC and its users would be in the same vicinity).

5) Individual performance evaluation: In the previous discussion, we view the decentralized cloud as a society and evaluate the performance from the perspective of social welfare. In this subsection, we evaluate the performance from an individual SDC's point of view. We randomly assign the workload arrival models for each SDC. There are 7 SDCs (1,3,4,7,11,17,22) with the Poisson arrival, 9

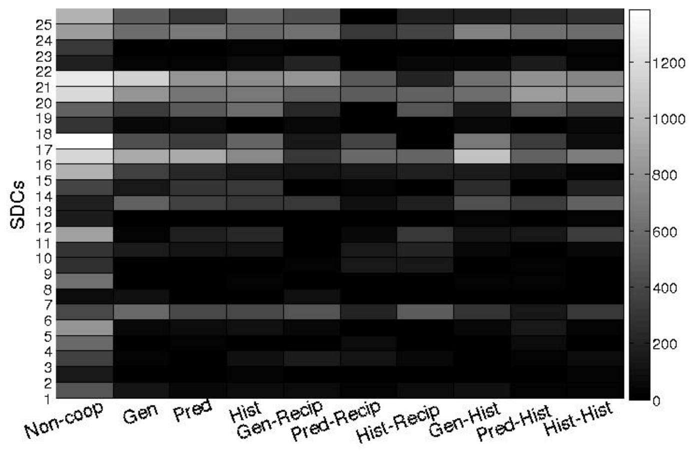

Fig. 9: Individual SDC performance

SDCs (2,6,8,9,10,13,14,21,24,25) with MMPP and 8 SDCs (5,12,15,16,18,19,20,23) with heavy-tailed arrival. Figure 9 shows the performance of each individual SDC under different strategies. The color of each grid, from black to white, in Figure 9 represents the request loss of an SDC i.e., more black means less loss. We can either compare the performance of different strategies by column or compare the performance of different SDCs by row. From the results, several interesting observations can be made: 1) the first column represents the performance in non-cooperative environment, which has the worst performance due to no resource sharing. The columns with reciprocity-based strategies are better than other columns since the color is almost entirely covered by black. Especially the column with Pred-Recip strategy performs generally well no matter what the workload arrival. 2) With the adoption of one appropriate strategy, any SDC in the decentralized cloud can improve their performance. In the Figure 9, we can always find a strategy for an SDC to suffer less loss than that in the first column which is the worst performance due to non-cooperation. For example, the best choice for SDC 21 is Hist-Recip while that for SDC 16 is Gen-Recip. This means it is necessary for the SDCs to learn that how to discover the appropriate strategy under different conditions. 3) If we use the difference between the best and the worst performance to evaluate the incentives of the SDCs, we find out the incentives of different SDCs for cooperation in decentralized cloud vary a lot. For example, for SDCs 2 or 7, there is no striking improvement for them to participate in the decentralized cloud. Since they find little incentive to cooperate, they may leave the decentralized cloud. While for SDC 17 and 21, there is a strong incentive to cooperate with others since their request loss can reduce dramatically through cooperation. If we take a further step to correlate incentive with other features of the SDC, we find that the variance in incentives has weak correlation with the SDC's capacity or workload type. This is because we observe that most of the SDCs with weak incentives have dissimilar workload type or capacity. Given that both the arrival rate and the workload execution time also have the same average, we infer that there is no predominant factor to determine the incentives of the SDCs to participate in the decentralized cloud. Therefore, there will be many SDCs, *regardless of capacity or workload type*, which will join the

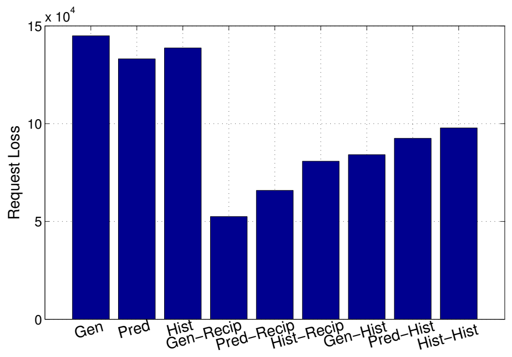

Fig. 10: Performance with a simulation time of 1000

decentralized cloud and achieve better performance through cooperation.

6) Long-scale simulation: In this part, we extend the simulation to a duration of 1000 to observe long term performance of different strategies. If 1 time unit is assumed to be 1 hour, the duration of simulation is about 42 days. The results are shown in Figure 10. Again, we observe that reciprocity-based strategies outperform the others. Meanwhile, it is interesting to see that the Gen-Recip is the best one rather than Pred-Recip as in Figure 7. We infer that in the long run, the performance of prediction-based strategy largely depends on the prediction precision. As discussed earlier, in the real cloud scenario, it could be very difficult to predict demand and therefore the losses incurred through misprediction will accumulate as time goes by. And this is exactly what our results show.

However, it is worthwhile to note that Gen-Recip based strategy has a good performance, which almost reduces the request loss by over 55% as compared to the General strategy. Moreover, it gives us two important implications: 1) We observe that the Gen-Recip based strategy is better than those based on history and prediction. That means it is better for SDCs to be generous to their partners rather than to judge them based on history or prediction. We believe that in the real world cloud scenarios, most of the SDCs cannot be fully rational since they do not have complete information to perform a perfect evaluation of their partners' requests based on history or prediction. It is better to keep it simple, and to offer help to their partners as long as they can. 2) We find out that as time passes, the optimal strategy for SDCs will change. In a decentralized cloud with high heterogeneity, the optimal strategy for each SDC may vary. To derive the optimal strategy, the SDCs have to employ an ongoing process of learning.

C. Competition of Strategies in a Heterogeneous Setting

In previous discussion, all the SDCs in the decentralized cloud adopt the same strategies. In this subsection, we will simulate a decentralized cloud of 25 SDCs with heterogeneous strategies. The goal is to discover which strategy can thrive in a heterogeneous environment. In this set of experiments, we exclude three strategies without PM and only consider six strategies based on reciprocity and history with PM. The strategy of each SDC is assigned randomly but the parameters of each type of strategy are the same, with risk indicator as 0.6, altruism level as 0.1 and tolerance as 1.

In this simulation, we design an adaptive strategy selection algorithm which is similar to the selection in genetic algorithms. However, we do not do explicit population wide selection, since in our case different SDCs can have different resources (as in capacity) and different needs (location and privacy concerns). Therefore, each SDC is interested in the strategy that works best for it. Our selection algorithm has two phases: training and testing. The time length of two phases are the same. Moreover, for each SDC, we define a *fitness function* to evaluate the performance of each strategy. Throughout the training phase, each SDC will change its strategy to a new random strategy at regular intervals. In other words, if we set the length of training phase to be 300 time units, the expected training time of each strategy is 50 time units in each training phase. At the end of training phase, the fitness function will compare the request loss of each strategy and return a strategy with the minimal request loss. After that the SDC will use this selected strategy without changing for the entire testing phase. The SDC will compute the total loss in the both training $loss_{train}$ and testing phase $loss_{test}$. It is clear that if the selected strategy under testing is the optimal one, the loss in the testing phrase must be less than that in the training phrases. Thus, in the end of the testing phase, the SDC will compare the loss of two phases. If $loss_{test}$ is greater than $loss_{train}$, it will switch to training phase to find other potential better strategies. Otherwise if $loss_{test}$ is less than $loss_{train}$, it will proceed with the next testing phase to continuously verify the selected strategy. This algorithm simulates the dynamics in the decentralized cloud in a random way and learns the performance of each strategy in a trial-and-error manner. This process will iterate until the end of simulation. We set the

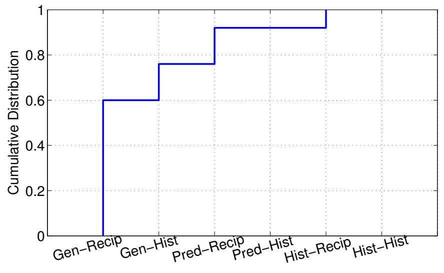

Fig. 11: Competition of strategies

length of both training and testing phases as 200. During the training phase, we change the strategy every 10 time units, and the entire duration of simulation is 1800 time units. Figure 11 shows that 60% of all 25 SDCs choose the Gen-Recip strategy as their best choice while the total number of Reciprocity-based strategies takes up 84% with only 4 SDCs adopting Gen-Hist strategy. It is interesting to observe that the experiment also simulates the survival of the fittest, which both the Pred-Hist and Hist-Hist strategies cannot survive during the competition since no SDC adopts those two strategies. We also observe that three of the four SDCs that choose the Gen-Hist strategy have heavy-tailed arrivals. We infer that history plays a more positive role in finding better partners for the SDCs with high burstiness (as in the case of heavy tail arrivals). Combining those results with the ones in Figure 10, it is safe to conclude that, in the long run the reciprocity-based strategy is the most effective strategy under varying conditions. We also increase the interval of changing a new strategy to 20 and 30 time units to observe the effect of longer running times on the fitness of the strategies, and observe similar results.

D. Discussion

Based on our extensive simulation of realistic decentralized cloud, we can answer the three questions raised in the beginning. To overcome the resource provisioning dilemma through cooperation in the decentralized cloud, different SDCs can employ a wide range of more or less sophisticated strategies. We observe that the performance of history-based strategies depends on the altruism level while that of prediction-based strategies on the risk indicator. For history based strategies, a strategy with high performance is the one that encourages selfless behavior, while for the prediction based strategies, the one with moderate risk evaluation is successful. Also, we discovered that the reciprocity-based strategies are the simplest yet the most effective way to develop cooperation in a variegated decentralized cloud. Compared to other strategies, the reciprocity-based strategies can immediately produce both cooperation and retaliation for their partners, rather than relying on history or prediction. This is a counter-intuitive result since on first thought, it would appear that prediction analysis applied on historical data would allow for better performance.

Furthermore, in our competition setting, we observe that the reciprocity-based strategies can also thrive in a dynamic environment with different strategies. The competition models the learning process of an SDC, through trial and error, to reach the optimal strategy. Through many trials, the SDCs can find the fittest strategy eventually. However, this process of learning might take a long time to move slowly toward mutually rewarding strategies. In the real world, the cost of learning for SDCs is high and they may not have enough patience to try. Thus, our results pave the way for the SDCs to speed up the learning process.

Nevertheless, our model is not without its limitations. First, validating our model with realistic workload dataset rather than synthetic ones would give us more confidence in our results. Second, even though we consider some intuitive strategies, the design space of the strategies can be huge (e.g., strategies that capture time of day correlations *or* distribution of request demand across multiple partners instead of rejecting the request if one partner cannot satisfy it fully, as in our current model). Thus we cannot be sure of the goodness of our winning strategies in face of unknown strategy variants. Finally, in our model, we mostly focus on the request loss of the SDCs while a more general model shall incorporate other factors related to the QoS.

VII. Conclusion and Future Work

This paper has proposed a decentralized cloud model in which a group of networked SDCs can work collaboratively to overcome the limitations raised by massive centralized cloud infrastructure. We also present a general strategy function to evaluate the performance of cooperation based on different dimensions of resource sharing. Our results show that the reciprocity-based strategies are more effective than other strategies, which can help the SDCs improve the performance of cooperation. In the future, we are interested in comparing our model with the one based on monetary payment. Furthermore, we also plan on evaluating our approach using real workload datasets.

Acknowledgment

This research is funded by the EU project CloudSpaces: Open Service Platform for the Next Generation of Personal clouds (FP7-317555). We also thank our shepherd Dr. Stefan Schmid for his insights during the last phase of our work.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, “A view of cloud computing,” *Communications of the ACM*, vol. 53, pp. 50–58, 2010.

[2] P. De Filippi and S. McCARTHY, “Cloud computing: Legal issues in centralized architectures,” in *VII International Conference on Internet, Law and Politics*, 2011.

[3] “European and german privacy laws,” http://www.martindale.com/business-law/article_Mayer-Brown-LLP_1294608.htm.

[4] “Business cluster,” http://en.wikipedia.org/wiki/Business_cluster.

[5] M. Van Alstyne, E. Brynjolfsson, and S. Madnick, “Why not one big database? principles for data ownership,” *Decision Support Systems*, vol. 15, no. 4, pp. 267–284, 1995.

[6] A. Chandra, J. Weissman, and B. Heintz, “Decentralized edge clouds,” *Internet Computing, IEEE*, vol. 17, no. 5, pp. 70–73, 2013.

[7] O. Babaoglu, M. Marzolla, and M. Tamburini, “Design and implementation of a p2p cloud system,” in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 2012, pp. 412–417.

[8] N. Samaan, “A novel economic sharing model in a federation of selfish cloud providers,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013.

[9] H. Wang, F. Wang, J. Liu, and J. Groen, “Measurement and utilization of customer-provided resources for cloud computing,” in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 442–450.

[10] I. Petri, T. Beach, M. Zou, J. Diaz-Montes, O. Rana, and M. Parashar, “Exploring models and mechanisms for exchanging resources in a federated cloud.”

[11] W. Wang, D. Niu, B. Li, and B. Liang, “Dynamic cloud resource reservation via cloud brokerage,” *University of Toronto, Tech. Rep*, 2012.

[12] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres *et al.*, “The reservoir model and architecture for open federated cloud computing,” *IBM Journal of Research and Development*, vol. 53, 2009.

[13] T. Locher, R. Meier, R. Wattenhofer, and S. Schmid, “Robust live media streaming in swarms,” in *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*. ACM, 2009, pp. 121–126.

[14] R. M. Axelrod, *The evolution of cooperation*. Basic books, 2006.

[15] K. Wang, M. Lin, F. Ciucua, A. Wierman, and C. Lin, “Characterizing the impact of the workload on the value of dynamic resizing in data centers,” in *INFOCOM, 2013 Proceedings IEEE*, pp. 515–519.

[16] H. Li, M. Muskulus, and L. Wolters, “Modeling job arrivals in a data-intensive grid,” in *Job Scheduling Strategies for Parallel Processing*. Springer, 2007, pp. 210–231.

[17] S. Mirtchev and R. Goleva, “Evaluation of pareto/d/1/k queue by simulation,” 2008.

[18] R. Rahman, T. Vink, D. Hales, J. Pouwelse, and H. Sips, “Design space analysis for modeling incentives in distributed systems,” in *ACM SIGCOMM 2011*. ACM, 2011.

[19] M. Feldman and J. Chuang, “Overcoming free-riding behavior in peer-to-peer systems,” *ACM SIGecom Exchanges*, vol. 5, pp. 41–50, 2005.

[20] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, “Karma: A secure economic framework for peer-to-peer resource sharing,” in *Workshop on Economics of Peer-to-Peer Systems*, vol. 35, 2003.

[21] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, “The eigentrust algorithm for reputation management in p2p networks,” in *Proceedings of the 12th international conference on World Wide Web*. ACM, 2003, pp. 640–651.

[22] A. Rapoport, *Prisoner's dilemma: A study in conflict and cooperation*. University of Michigan Press, 1965, vol. 165.

[23] C. Dana, D. Li, D. Harrison, and C.-N. Chuah, “Bass: Bittorrent assisted streaming system for video-on-demand,” in *Multimedia Signal Processing, 2005 IEEE 7th Workshop on*. IEEE, 2005, pp. 1–4.

[24] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, pp. 23–50, 2011.