Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

KeyPIn – mitigating the free rider problem in the distributed cloud based on Key, Participation, and Incentive

Doyel Pal[1*], Praveen Khethavath[1], Johnson P. Thomas[2] and Utpal Mangla[3]

Abstract

In a distributed cloud, unlike centralized resource management, users provide and share resources. However, this allows for the existence of free riders who do not provide resources to others, but at the same time use resources that others provide. In a distributed cloud, resource providers share resources in a P2P fashion. In this paper, we propose a 3-pronged solution KeyPIn—a Key-based, Participation-based, and Incentive-based scheme to mitigate the free rider problem in a distributed cloud environment. We propose an incentive-based scheme based on game theory for providers to participate in the cloud by providing resources. This participation will be low for free riders thereby limiting their access to resources. A secure time instant key is generated based on a key management scheme that enables good users' more time to access resources as their participation is high, whereas free riders are given limited or no time as their participation is low. Simulation results show that our scheme is effective in mitigating the free rider problem in the distributed cloud.

Keywords: Distributed cloud, Free rider, Game theory, Trust, Participation, Key, and Incentive

Introduction

Cloud computing refers to delivering configurable computing and network resources over the internet to users on-demand. It uses virtualization techniques [1–3] to provide required resources to users dynamically in the form of virtual machines (VMs). Cloud providers provide resources in the form of various instances of platform, infrastructure, and storage as services to users. For example, Amazon [4] provides different sizes of virtual machine instances – Small, Medium, Large and Extra Large. Cloud Computing use massive datacenters which lead to communication overhead as the number of users increases. Moreover, enterprise usage of the cloud is much higher when compared to individuals' usage of the cloud. A large percentage of individuals use the cloud for sharing and storing data free-of-charge. Cloud computing-based traffic has increased significantly, and global cloud datacenter traffic is forecasted to an estimate of 19.5 Zettabytes per year by 2021 by Cisco [5]. Cloud accounts for 95 percent of global data center traffic [5]. Most of the cloud service providers such as Google, Amazon, Facebook, and many others are expanding and building new datacenters worldwide.

There is a huge increase in demand for cloud resources. Cloud computing is made possible because of virtualization technology. By moving towards cloud-based services many resources such as Desktops, PC's and servers which can run virtualization software and can create multiple VMs on them are not being used to their fullest capabilities [6]. A distributed cloud developed using the resources provided by individual resources can mitigate the disadvantages of cloud computing. Moreover, it will be helpful to reduce internet traffic by reducing the load on cloud data centers. A distributed cloud makes use of unused individual

*Correspondence: dpal@lagcc.cuny.edu
[1] Mathematics, Engineering and Computer Science Department, LaGuardia Community College, CUNY, Long Island City, New York City, NY 11101, USA
Full list of author information is available at the end of the article

resources, which are not used to their maximum capacity and thereby avoiding investments on new data centers.

In the distributed cloud users, both provide resources and use resources. Resource providers are distinctive i.e., these distributed cloud servers are offered by individuals with resources to offer. Resource requestors may request resources from other users in the distributed cloud who then become resource providers. Users of the distributed cloud must discover these resource providers and request the resources. A distributed cloud uses a decentralized mechanism to discover and allocate resources where users share recourses in a P2P fashion [7]. In our model resources are shared by users, but there might be some users, so-called "free riders", who do not provide any resources for others, but use the cloud to get resources for themselves. Free riders should be given limited access or prevented from getting access to resources. This could be achieved by exacting a high cost to free riders for using the system. Hence, the system must be stable after resource allocation. In our model, a system is said to be stable when the resource allocation is fair, that is, users who provide resources for other users are more likely to get the resources they require.

Our contributions are as follows:

1. An incentive-based scheme is proposed where users who provide resources (the resource providers) to other users (the resource requestors) are assigned a higher participation factor (ρ) and free riders will have a lower participation factor. Hence users have an incentive to provide resources whereas free riders will be denied or given very limited resources.
2. In addition to providing incentives, free riders must be identified so that they have limited access to resources in the cloud. The participation factor over all nodes in the cloud is proposed to identify free riders based on our proposed incentive-based scheme.
3. To control resource access to requestors, a time-based access scheme based on participation is proposed. A resource requestor with a high participation value is given more time to access resources when compared to a free rider who will have a lower participation value and is thereby given less time to access resources.

The problem specification is presented next. This is followed by the literature review and preliminaries (distributed cloud architecture). Next, we propose an incentive-based scheme to identify resource providers and motivate them to provide resources to genuine users. Also, we present a trust scheme to detect free riders and the time-based access scheme to share resources. Simulation results are presented next before the paper concludes.

Problem specification

In research [8] conducted on P2P networks, researchers found that the percentage of free riders is around 85% of all users. Statistics show that 99% of resources in Gnutella [8] is offered by only 25% of peers, i.e., the remaining 75% of users provide only 1% of the resources. The free rider problem is therefore a serious issue and there needs to be a mechanism which addresses this potential problem. Mitigating the free riders' problem is very challenging and important as the existence of free riders clog the network and hinder the growth of the network. In this paper, we propose an incentive-based mechanism based on game theory and use this to mitigate the free rider problem in a secure manner.

Literature review

All peers are expected to share their resources in a peer-to-peer network. Free riding is a genuine concern in a network since free riders are reluctant to share their resources with their peers. As mentioned in [8] up to 85% of users may be free riders. Existence of free riders in a network affects the system in multiple ways such as, CPU overloading, unnecessary traffic in the network, under-provision of network resources, single-point-of-failure, degrading system utility, system collapse etc. [9–11]. As free riders do not share their resources, honest or good users in a network face difficult tradeoffs between sharing resources and not sharing resources. In the distributed cloud, resources are provided by users in a P2P manner [12]. Resource providers (RPs) in a distributed cloud architecture also suffer from the free rider problem when any user becomes reluctant to share resources with others.

Free riders clog the network and hinder the growth of the network. Solutions to reduce free riding problems can be categorized into three groups – monetary-based, reciprocity-based, and reputation-based approaches [11, 13]. The monetary-based approach charges peers for any received service. The accounting module of monetary-based approach securely stores each peer's virtual currency, and the settlement module exchanges the virtual currency for services. For a current session, the reciprocity-based approach monitors peers' contributions to assess peers. It does not keep track of long-term history of peers and that lets a peer be judged as a free rider in one session and contributor in the next. Assuming peers report their interaction honestly, the reputation-based approach keeps track of long-term behavior of a peer to measure peers' reputation. Some implementation issues with this approach are reputation reliability,

communication overhead, persistent identifiers. In this paper, we focus on reputation and trust-based approaches in the distributed cloud to mitigate the free rider problem because the distributed cloud is completely decentralized, and peers need to be both trustworthy as well as reputed to be reliable and successful. When applied to P2P networks, the monetary-based approach has several implementation limitations such as centralization and communication overhead, persistent identifiers, mental transaction costs. The reciprocity-based approach has several implementation issues such as fake services, peer identity management, and contribution-level credibility. In [12, 14–16], authors proposed trust-based approaches in different networks. In [12] the authors proposed a trust model based on human cognitive behavior and incorporated multiple trust factors to reflect the complexity of trust. For social networks, a game theory-based trust measurement model based on service reliability, feedback effectiveness and recommendation credibility has been proposed in [14]. To alleviate the free rider problem, the authors have proposed a game theory-based punishment mechanism for global trust and punishment cycles for specific trust. A secure, robust Reputation and Risk evaluation-based Trust management framework named R2 Trust [15] uses both reputation and risk to evaluate the trustworthiness of a peer. In ubiquitous and pervasive computing environments, Trust computation and management system (TOMS) based on trust management has been proposed by the authors of [16]. The trust management of TOMS makes decisions on nodes' access rights by developing a trust model, assigning credentials, and managing the trust value of each node and updating private keys. The way resources are used and provided in P2P computing and distributed cloud are completely different. The distributed cloud uses an additional layer on top of P2P to perform resource discovery and allocation. In the distributed cloud unlike P2P networks, the provider has control over how much of his resources can be made available and at the same time the user has the choice of choosing a provider from a pool. This poses a completely new challenge for evaluating reputation and trust and therefore a need for a different mechanism to evaluate trust and reputation.

Preliminaries

Distributed cloud architecture

With the advent of cloud resources, a distributed cloud model [7] was created by considering a huge number of resources that are not being utilized to their full potential. 72.9% of the cloud services were used for enterprise solutions and the remaining were used by consumer services which includes Gmail, Twitter, Facebook, YouTube, Dropbox, Google Drive, and many others. According to Cisco, consumer IP traffic is much higher than business IP traffic [5]. Currently, this traffic is directed to and from data centers which results in the creation of single point bottlenecks.

A distributed cloud model is shown in Fig. 1. In a distributed cloud, resources are discovered by users [7]. The nodes in distributed cloud model can be both *RPs and RRs*. All nodes in this model can communicate with each other in a P2P fashion. There are different factors that need to be considered when discovering resources, such as latency, throughput, locality etc. Instead of discovering

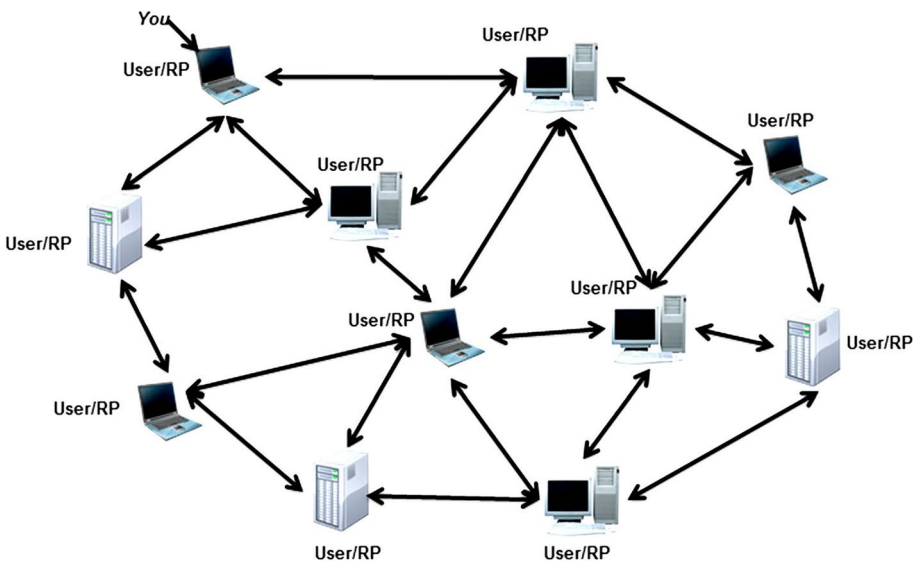

Fig. 1 Distributed Cloud Model

only the exact number of resources requested by a user, n number of resources are selected in an optimized manner to minimize latency or maximize throughput. As a result, resources that fit user requirements and have low latency or high throughput are chosen to ensure efficient resource allocation in the distributed cloud.

In the distributed cloud, all the resources are shared in a P2P fashion. In the distributed cloud, users are not charged to use the cloud. Instead, they are expected to provide resources to other users. Free riders use the resources of other users in the distributed cloud without providing their own resources for others to use. Each node in the distributed cloud will run a distributed cloud overlay application over a P2P architecture.

Providing resources to users

Resource discovery

A user requesting resources (or Resource Requestor *RR*) must first discover the resources. We proposed a scheme to discovery resources using a modified Kademlia [17] protocol to perform resource discovery. Kademlia is a P2P, decentralized protocol used to identify peers by making use of DHT's. Kademlia uses bitwise XOR metric to perform accelerated lookups and is shown to perform efficient query lookups. In a network of size N, locating a node takes an average of O(log$N$) hops. To perform resource discovery, a multi-valued hash table scheme was used [7]. In this scheme, we proposed a concept of distributed local multi-valued hash table to identify resources using a range of attributes. Each node in the distributed cloud is given two ID's, one is to locate the node and other is to identify attributes of a node. Each node has different attributes that are used for the resource discovery mechanism. This scheme is implemented by modifying the Kademlia protocol.

Incentive for providing resources

The distributed cloud does not charge users to use the cloud apart from providing resources to other users. Free riders are users who use the resources of other users in the distributed cloud without providing their own resources for others to use. A critical issue is how much resources a resource provider should allocate to a user or resource requestor given there are free riders in the network. After discovering resources, a game theoretic approach is used to allocate appropriate resources to requestors offered by *RP*s. In this section we propose an incentive-based scheme that will give more resources to genuine or good users and provide fewer resources to free riders or bad users.

A *utility function* (μ) determines the utility which is the value of resources assigned to the user. This is based on the participation factor of a user. A participation factor ρ_y is assigned to each user y. The participation factor of a node y in the distributed cloud is a measure of the value of node y's resources that y has made available to others. In our mechanism a user x requesting resources (or resource requestor *RR*) receives bids from *RP*s and then chooses the *RP*s rationally. Users (or resource requestors *RRs*) with low participation factors will receive less time to access resources. Users need to provide resources, to raise their participation factor. The participation factor thus serves as an incentive to encourage users to provide resources. Hence, there are two types of players, a Resource Provider (*RP*) of resources and a user of resources who requests the resources (*RR*).

During the initial stages or if the user's ρ is low, he might receive resources whose utility value is small. Once the system is close to stability, users who are not free riders will most likely receive resources with maximum or close to maximum utility. Ideally each provider will provide resources to maximize her own participation factor. In the distributed cloud, users will be requesting multiple resources for computation, so a simultaneous auction with multiple winners is used. A bidding profile is a vector of player i's bids $b_i = b_1, b_2 ... b_{\mathrm{i}}$. The bidding profile of user i is represented using b_{i} and the bidding profile of user i's opponent is defined as $b_{-i} = b_1..b_{i-1}..b_{i+1}b_{i+2}\ldots$ User i chooses a resource based on the utility, where utility μ_i is defined as:

$$\mu_i(b_i; b_{-i}) = f(MEM, CPU)$$

Utility μ_i is a function of memory, CPU. Once the user (or *RR*) receives all the bids from *RP*s, the user will calculate utility values for all the bids using the above equation. This game achieves Nash equilibrium. In $\mu_i(b_i; b_{-i}, \rho)$, b_i represents the winning bid and b_{-i} represents all the bids that were lost.

In practice, RR is bargaining with multiple providers to get the best deal. This bargaining game is played in rounds. In each round, the user *RR* makes a request, and the provider *RP* decides to accept it or not. Acceptance ends the game while rejection leads to the next round. A strategy s_i of player i is a function that assigns an action to player i when it is its turn to move. As a notational convention, $-i$ represents the player other than player i in the bargaining game. Similarly, s_{-i} represents the strategies of the player other than player i. Note that $s = (s_i, s_{-i})$ is a strategy profile.

A user who needs resources, that is the *RR*, makes the first move. A user gains access to resources he has requested to accomplish his task. This may be Memory, CPU etc. Some tasks are important whereas others are less important. There is therefore a weight α associated with each request. This is labeled as $\alpha(CPU, Mem)$.

CPU and Mem resource parameters are normalized to between 0 and 1.

Each round of the bargaining process involves a cost c for the users, both *RR* and *RP*. The cost for a *RR* may involve, for example, acquiring and managing resources provided by *RP*s. The total cost is labeled as $\sum_{i=1}^{m} c_{Ri}$ for m rounds. The cost for a *RP* includes unavailability of resources during the time the resources are being used by a *RR*, for example. The total cost is labeled as $\sum_{i=1}^{m} c_{Pi}$ for m rounds.

RP makes the next move. *RP* provides resources to *RR* to accomplish the requested task of *RR*. This is labeled as $\beta(CPU, Mem)$. β is a weight, indicating the loss factor. A resource provider who has a lot of resources but contributes little will have a low β value, whereas a *RP* with few available resources but contributing proportionally a larger share will have a high β. β is a measure of the loss to the *RP* in providing the resources to the *RR*.

The utilities μ_U and μ_{RP} of user *RR* and *RP* are therefore:

$$\mu_{RR} = \alpha(CPU, Mem) - \sum_{i=1}^{m} c_{Ri} \tag{1}$$

$$\mu_{RP} = -\beta(CPU, Mem) - \sum_{i=1}^{m} c_{Pi} \tag{2}$$

In this game, we assume that this is a full information game, that is, *RR* has full information about *RP* and vice-versa.

A bargaining game is a special case of extensive game with perfect information [18]. In an extensive game with perfect information Γ, a history h is a sequence of actions starting from the beginning of the game. A subgame is the remaining part of the game following a specific history. Denote by $\Gamma|h$ the subgame that follows the history h. Let $s_i|h$ denote the strategy that s_i induces in the subgame $\Gamma|h$, and $u_i|h$ denote the utility of player i in subgame $\Gamma|h$. In extensive games, an important solution concept is sub-perfect equilibrium [18].

Definition 1: A subgame perfect equilibrium of an extensive game with perfect information Γ is a strategy profile s^* such that for every player $i \in N$ and every nonterminal history h, after which it is player *i's* turn to take an action, we have.

$$u_i|_h(s_i^*|_h, s_{-i}^*|_h) \geq u_i|_h(s_i, s_{-i}^*|_h)$$

for every strategy s_i of player i in the subgame $\Gamma|h$.

The game studied in this paper has a finite horizon, which means that the number of rounds is finite and the number of actions at any round is finite. To verify a strategy profile s^* is a subgame perfect equilibrium in a game with a finite horizon we use the one deviation property [19]:

Lemma 1: (The One Deviation Property) [20]: *The strategy profile s^* is a subgame perfect equilibrium of a finite horizon extensive game with perfect information Γ if and only if for every player $i \in N$ and every history h, after which it is player i's turn to take an action, we have*:

$$u_i|_h(s_i^*|_h, s_{-i}^*|_h) \geq u_i|_h(s_i, s_{-i}^*|_h)$$

for every strategy s_i of player i in the subgame $\Gamma|h$ that differs from s^* only in the action it prescribes after the initial history of $\Gamma|h$

Non-incentive scheme

We now show why free riders exist in the cloud. *RR* makes the first move.

Theorem 1: *There is no incentive for the resource provider RP to provide any resources as RP's utility μ_{RP} is always less than* 0, *that is, it is always negative.*

As the *RPs* utility is always negative, the resource provider starts acting selfishly or a free rider. Hence *RP* will not provide any resources but may request for resources.

Proof: Follows from (1) and (2). *RP* always has a loss when allowing a *RR* to access its resources.

To avoid a resource provider from becoming a free rider, we propose an incentive-based scheme.

Incentive-based scheme

An incentive is therefore provided to the *RP*. The incentive is the participation factor ρ which is a measure of the resources a node or user has provided to others in the cloud. A user with a high ρ is more likely to provide resources to other users than a user with a low ρ. Furthermore, a *RP* is more likely to provide resources to a *RR* with a higher ρ as this means the *RR* is more likely to provide resources. The main idea is to influence the players strategy by introducing an incentive. This incentive, the participation factor ρ to the provider is therefore added to the utility function. The advantage of introducing the incentive is that we can influence the provider's best strategy in the game, such that the offer in the first round is a "reasonable" price for the resources. This price is "reasonable" in the sense that it makes the transaction profitable for both parties. That is, both parties will have positive utilities in the game. Furthermore, accepting this "reasonable" price is also to the best interest of the

provider. The requestor or user of resources maintains the same utility when he uses resources, and the provider adds ρ to her utility when she provides resources. ρ does not increase for a node using resources, whereas it does increase for a node providing resources. A requestor may provide resources and become a *RP* and similarly a *RP* may become a *RR*. Hence for the stability of the system the participation factors must be at an acceptable level. The utilities of RPs and RRs are now therefore:

$$\mu_U = \alpha(CPU, Mem) - \sum_{i=1}^{m} c_{Ri} \tag{3}$$

$$\mu_{RP} = \rho - \beta(CPU, Mem) - \sum_{i=1}^{m} c_{Pi} \tag{4}$$

RP has an incentive ρ to provide resources where ρ_j is the participation factor of a user *j*. The participation factor increases as *RP* provides more resources. In the notation below, a subscript denotes a particular participant in the cloud as a resource requestor or a resource provider. $[\alpha(CPU, Mem) - \sum_{i=1}^{m} c_{Ri}]_k$ denotes utility of a requestor k and $[\rho_k - \beta(CPU, Mem) - \sum_{i=1}^{m} c_{Pi}]_k$ denotes utility of a provider k where ρ_k is the participation factor of k.

Analysis of the scheme

We use backward induction as we start at the end of the bargaining process. The last mover in the game is *RP*. In the last *m*th round the payoff for *RP* will be as in (4). For the *RR*, the payoff will be as in (3). Given the incentive ρ, the system will choose the resource provider who provides the closest amount of resources to that requested by the requestor.

$$argmin\left(\left[\rho_k - \sum_{i=1}^{m} c_{Pi} - \beta(CPU, Mem)\right]_k - [\alpha(CPU, Mem)]_j\right)$$

The cost of bargaining to RR_j is not considered as it has no effect on RP_k for the provision of resources by RP_k.

Since RP_k has provided resources its participation factor ρ^k will increase. This updated participation factor is considered when a provider *RP* becomes a requestor *RR*. The new participation factor of provider k in the n^{th} round is determined as follows:

$$\rho_k^n = \rho_k^{n-1} + f(CPU, Mem)$$

In other words, the new participation factor of RP_k is the previous participation factor added to a function of the resources k has provided in the current round.

In the $(m\text{-}1)^{th}$ round, the system will choose the resource provider who provides the closest amount of resources to that requested by the user which will be:

$$argmin\left(\left[\rho_k - \sum_{i=1}^{m-1} c_{Pi} - \beta(CPU, Mem)\right]_k - [\alpha(CPU, Mem)]_j\right)$$

Repeat until the first bargaining round. The lowest cost will therefore be at the first round, that is, when $m=1$. That is,

$$argmin\left(\left[\rho_k - c_{Pi} - \beta(CPU, Mem)\right]_k - [\alpha(CPU, Mem)]_j\right)$$

ρ_k will be different for each round. Since the minimum value for the participation factor ρ_k is in the first round we can say:

Theorem 2: *There exists a unique sub-game perfect equilibrium where the bargaining ends in one round where the resources that a resource provider will provide will be determined by*:

$$argmin\left(\left[\rho_k - c_{Pi} - \beta(CPU, Mem)\right]_k - [\alpha(CPU, Mem)]_j\right) \tag{5}$$

Proof: The proof is as outlined above using backward induction.

The above game is between one *RP* and one *RR*. The game will be played between one *RR* and many *RP* s. Hence there are many simultaneous games where each game is between one *RR* and different *RP* s. Since they are independent games, the above equations apply to each game and *RR* will choose the *RP* with the minimum price. The bargaining ends in one round. The provider that provides closest to the requested resources is chosen. Hence participation factor is not simply a matter of providing the most resources. It is a based on providing only the required amount of resources or as close as possible to the requested amount (hence the *argmin* function). Providing the most resources will therefore not satisfy (5) above. In the proposed scheme, a selfish provider may not get all the resources he needs as he will have a low participation factor. A *RR* who has not provided any resources will not be able to participate in the cloud unless they start to provide resources. Moreover, in the proposed scheme the incentive will be the minimum amount as in (5) and higher incentives are not required.

Detecting free riders in the distributed cloud

To identify a free rider, we calculate the participation factor of all participating users in the distributed cloud. As mentioned above, the participation factor of a user is a measure of his resource contributions to the distributed cloud. A high participation factor means that the user

provides a lot of resources to the cloud. A participation factor is the value that a resource provider RP_x calculates for a particular requestor of resources RR_y based on its behavior. A participation factor is the collection of experiences that all *RPs* (other than RP_x) have on a particular resource requestor RR_y. Each *RP* determines the participation factors of the *RR* who has requested its resources as shown in Eq. (6). In addition, a RP_x calculates the participation factor of a resource requestor RR_y by the participation factor values that other RPs have of RR_y as shown in Eq. (7) and Eq. (8). A participation factor value less than a threshold trust value determines a user as a free rider. Although RR_y may be able to access resources, it will be allowed to access these resources only for a time determined by the participation factor. Hence free riders will be given less time to access resources of a provider when compared to a requestor who is not a free rider. Therefore, the free riders' problem will be mitigated in a distributed cloud network. We next look at how legitimate users can request and use resources from other *RPs* securely based on their participation. Free riders or requestors with poor participation on the other hand will get minimal access time to use resources from a provider in the distributed cloud. Access time of a user depends on their pattern of behavior (or participation factor) with other *RPs*, that is, if they are good users or free riders.

Alleviating the free rider problem in the distributed cloud is very important as free riders are reluctant to share resources. They use up resources freely, hence affecting the whole cloud system. We propose a novel mechanism to identify free riders in a distributed cloud network. A resource provider can also be a user who is requesting resources from other users. In other words, a resource provider (*RP*) may be a resource requestor (*RR*) at another time. Any requestor's behavior is determined by its participation factor since a free rider can be a good provider to a specific resource requestor but not provide any resources to other *RR*s. A server provides a time constrained key to the resource requestor *RR* who has requested resources and to the *RP*. When the time associated with a key expires, the user *RR* is denied access to the resource. This method is more applicable for users who are performing computation over the distributed cloud rather than using it for storage. This ensures that free riders with low participation factors will get minimal access to resources as their key will expire quickly, whereas good users with high participation factors will have sufficient time to access resources to complete their tasks.

Factors to determine free rider

From a practical implementation perspective, each resource provider should have at the minimum required capacity and CPU resources such that a minimum number of virtual machines (VM's) can be created. The minimum can be set by the system. The factors that determine the free rider are:

a. *Participation Factor* (ρ_y): The participation factor determines the behavior of users who are requesting resources, that is, resource requestors (*RR*). It is based on the number of requests that a node y which is requesting resources has fulfilled successfully in the past as a provider. The more the requestor has provided its resources in the past, the higher the participation factor ρ will be with $0 < \rho \leq 1$. The participation factor is a measure of how much resources a requestor has contributed to others in the system. In our model, each node x stores a value for ρ_y which is the participation factor that x has determined for node y (a resource requestor) based on previous requests it has made to y. A requestor or provider cannot set their own participation factor values and cannot change the values of other providers from whom it has received resources. The participation factor aims to prevent selfish behavior of users. The participation factor $\rho_{y,x}$ at a node x for a resource requestor RR_y is calculated as:

$$\rho_{y,x} = \sum_{i=1}^{t} \frac{N_i}{C_i} \tag{6}$$

N_i is a measure of resources provided by y as a resource provider during a request 'i' by requestor x and C_i is the capacity of available resources at y during request i, t is the total number of requests x has made.

However, this gives only a measure of the experiences x had with y. Node x, the RP may request other nodes (apart from the RR y) for their participation factor values of y. The overall participation factor of y in relation to all nodes in the distributed cloud is determined to be:

$$\rho_y = w_i \sum_{i=1}^{m} \frac{\rho_{y,i}}{g} + w_x \rho_{y,x} \tag{7}$$

where g is the total number of participation factor values received from other nodes, $\rho_{y,i}$ is the participation factor of y by i. w_i is the weight given and the sum of the weights must be 1. Higher weight is given to node x's own experience with y, that is w_x.

b. *Time-based participation factor* (TF_y) : The participation factor ρ as determined above, does not consider past historical behaviors, that is, it does not consider time. A resource requestor RR_i may have provided resources in the distant past but has been a free rider in the more recent past. On the other hand, a resource requestor RR_j may have been a free rider in the distant past but has provided resources in the more recent past. Both resource requestors RR_i and RR_j may therefore have very similar participation factors. However, RR_j should have a higher participation factor as its more recent behavior is a better indicator of its participation. Time based participation Factor for a resource provider RR_y by a node x is:

$$\rho_{y,x}^{(t)} = \alpha\rho_{y,x}^{(t-1)} + (1-\alpha)\rho_{y,x}^{(t-2)} \tag{8}$$

where $\rho_{y,x}^{(t)}$ is participation factor at the t^{th} request, that is, the latest request. $\rho_{y,x}^{(1)}$ is the first request. α is the temporal importance factor. If α is high, more weight or importance is given to more recent participation, whereas if α is low, more weight is given to older participation experiences.

Hence $\rho_{y,x}^{(t)}$ obtained in Eq. (8) is substituted for $\rho_{y,i}$ in Eq. (7) to get the final participation factor. The time t_x^y that a resource requestor RR_y has access to RP_x's resources is therefore a function of the participation factor of RR_y, that is,

$$t_x^y = f(\rho_y) \tag{9}$$

Additional factors such as quality of service provided by a resource provider could be included, for example, the time to access a resource. However, quality of service may be affected by factors outside the control of the resource provider, such as network traffic or congestion.

The interactions with a provider are kept in a table. Each resource requestor keeps track of the providers from whom it has received resources. The table will contain entries listing participation factors, number of interactions and measure of resources provided.

A good user with a high total participation factor gets more access time than one with a low participation factor, as a user with a low participation factor may be a free rider. A new node that enters the distributed cloud is given an initial value by the cloud system. This may be based on the average of the participation factors of all nodes, or a minimum value based on historical data.

Controlling resource access to resource requestor

Once the participation factor is obtained, the RP_x informs the RR_y of the participation factor. The resource requestor may get responses from no *RP*s or from many *RP*s. Each *RP* calculates the participation factor as outlined above. The responses from the *RP*s therefore may not all give the same participation factor. This is because the *RP*s do not have a global view of the distributed cloud. The *RR* selects the *RP* that satisfies Theorem 2, that is, Eq. (5). The resource requestor will determine the winner. The resource requestor RR_y informs the selected resource provider RP_x that it wishes to use its resources. RP_x has already calculated the participation factor of RR_y as determined by Eq. (8) because it had earlier informed RR_y of the participation factor.

Our goal is to propose a simple algorithm that is computationally not intensive to operate in the proposed distributed cloud. This is because the machines are commodity or off the shelf computers. The literature has reported few instances of time-based encryption [21, 22]. Time lock encryption has been proposed in [24] where a message is encrypted after a certain deadline has passed. This approach draws on bitcoin and time-lock puzzles. Our approach is based on [22] as it is a simple algorithm-based approach. Time is broken into time slices. If current time is t, for example, the time is determined to be $\lceil t \rceil$ where the time slice is pre-determined and constant. A key server generates a time sensitive key that will specify the time a requestor is allowed to access the provider's resources. The server takes as input a time $t_T = \left[\lceil t_s \rceil + \lceil t_x^y \rceil + \lceil n \rceil\right]$. t_s is the current time at *RP*, t_x^y is the time RR_y has to access RP_x's resources determined based on the participation factor calculated by RP_x (Eq. (9)), and n is a constant time that may be needed due to network and other delays. Our goal is to make sure that resources can be accessed by RR_y only within the time specified by the participation factor, that is, only for a time t_x^y. All messages are encrypted when communicating between (i) the server and nodes and (ii) between nodes. To ensure that keys cannot be reused, a session key is derived where this key can be used only for the current session for RR_y to access RP_x's resources. A different key will have to be generated for RR_y to access RP_x's resources in the future. The resource provider RP_x generates an access key (tok_m, a_{RR}) which RR_y will use to access RP_x's resources.

A public cryptography scheme is followed where the public key is $K_{Pi,}$ and the private key is K_{Ri} for a node *i*. The public key of the server is $K_{PServ,}$ and the private key is K_{RServ}. The server is assumed to be trustworthy. The protocol in Fig. 2 is outlined below.

Protocol:

1. Server generates Time Instant Key (TIK)

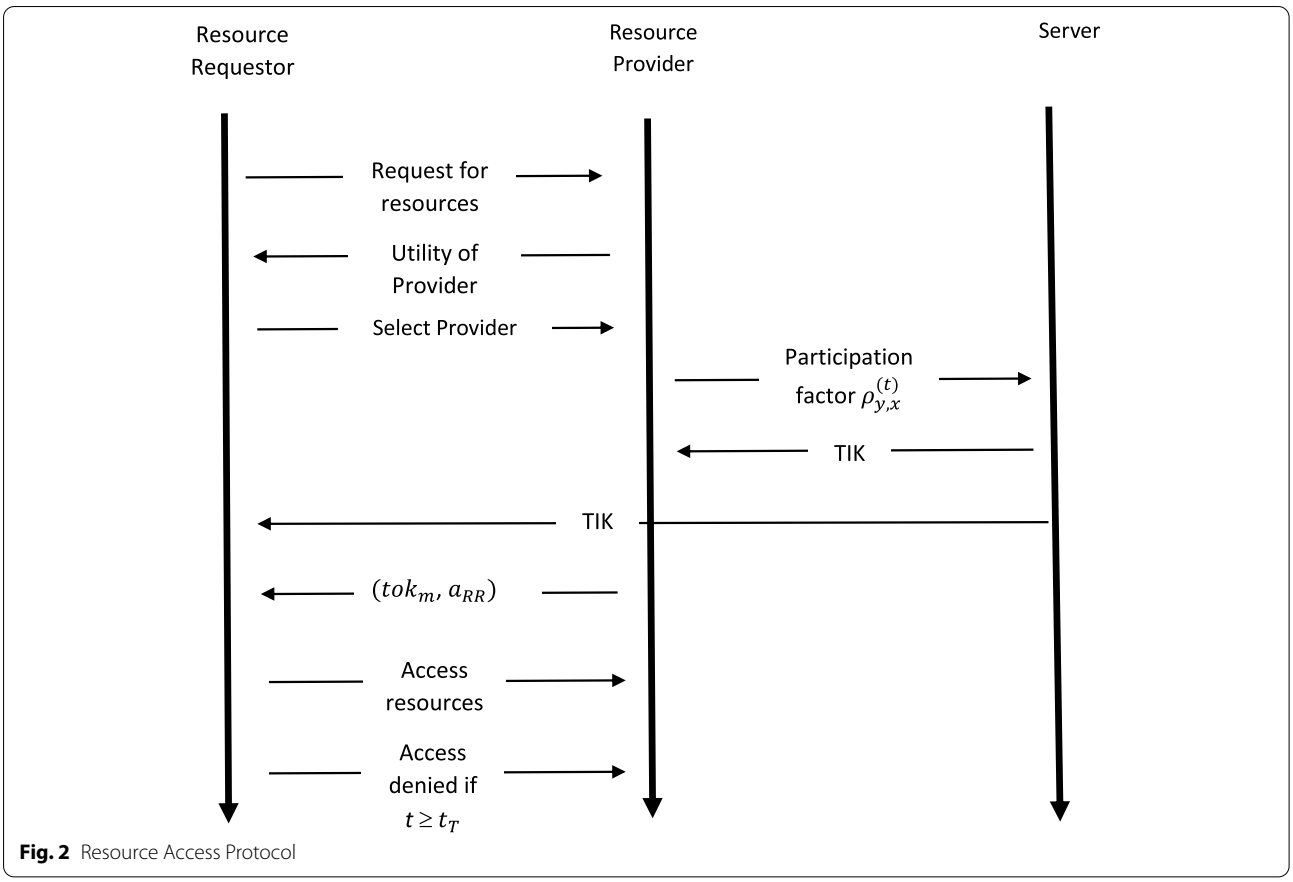

Fig. 2 Resource Access Protocol

Resource provider RP_x sends the server the participation factor (Eq. (8)) and the identity of the requestor RR_y, that is, $RP_x \xrightarrow{\rho_{y,x}, RR_y} S$ where S is the server. The server calculates the time t_x^y (Eq. (9)) that a resource requestor RR_y has access to RP_x's.

Using K_{Py}, K_{Px} where y is the requestor RR_y, x is the provider RP_x, S is the server and time $t_T = \lceil t_s \rceil + \lceil t_x^y \rceil + \lceil n \rceil$, the output is the Time Instant Key (TIK) k_t. k_t therefore includes t_T generated by the server, RR_y and RP_x.

2. Server distributes TIK to Requestor RR_y and Provider RP_x,
   The server sends the time key k_t to the Resource Requestor RR_y and the Resource Provider RP_x encrypted with their public keys. The time allocated to RR_y to access RP_x's resources can therefore be determined from k_t.
3. Resource Provider RP_x generates access key.
   Access rights tok_m for the resources requested by RR_y is generated by RP_x where $tok_m = (m_i, \wp(n))_{\{i \epsilon 11,\ldots,N\}}$. Here $\wp(n)$ is the power set of n. m_i is resource i and n is the set of all possible accesses on a resource (example: read access). This identifies the operations the requestor can execute on the *RPs* resources.
   An access code a_{RR} is also generated. The access code enables the access rights. Hence, when the time for the requestor to access RP_x's resources has expired, the access code is changed, thereby denying requestor RR_y to provider RP_x's resources. (An alternate approach would be to change the access rights to individual resources when the access time expires, but this would be more cumbersome). Using the shared symmetric TIK key k_t, the provider RP_x generates the access rights tok_m for the resources requested by the requestor RR_y and the access code a_{RR} to generate the access key ($tok_{m,}a_{RR}$). This ensures that when the access key is transmitted to the requestor RR_y, the access rights and access code cannot be read by an intruder who intercepts the message. If an attacker acquires access to the access rights and access code, he may be able to gain access to the provider's resources. Hence the access rights and access code are encrypted using the TIK key k_t.

It is also important to note that the secret TIK key k_t can be used only for this session of a provider providing resources to a requestor. Subsequent requests for resources will require a different secret key. This makes the scheme more secure than using the public keys of the provider and requestor as these can be used at any time and are not limited the current session.

4. Resource Provider RP_x sends access key to Resource Requestor RR_y.
   RP sends RR the access key (tok_m,a_{RR}) encrypted by the TIK key k_t
5. Resource Requestor RR_y extracts information in access key
   RR_y. will decrypt using the shared symmetric TIK key k_t since it was encrypted by k_t and RR already has k_t which was set by the server, to read the access token tok_m and access code a_{RR} to thereby gain access to the resources requested.
6. Resource Requestor RR_y accesses resources
   The Resource Requestor can access the resource only if $t \in [t_0,\ t_0 + t_T]$ where t_0 is the time the Resource Requestor RR_y first accessed the resource. The algorithm takes as input access rights tok_m, TIK k_t, access code a_{RR} and permits access to RP's resources if $t \in [t_0,\ t_0 + t_T]$, otherwise it returns a failure $\perp$.. t_0 is the start time of access. RR uses the access rights tok_m to gain access to the resources requested.
7. Resource Provider RP_x locks out Resource Requestor RR_y when $t \geq t_0 + t_T$

Decrementing counters are set by RP_x and RR_y which decrement with each time instant. Their initial values will be t_T.When $t_f = 0$, that is, $t \geq t_T$, RR_y is no longer permitted to access RP_x's resources. RP_x rekeys the access code a_{RR} for the resources that RR_y is using. That is, a_{RR} is changed to some other access code a_{RX} where no other requestors can use the resources of RP_x. Hence at the end of the time slice, RR_y will not be authenticated as the access code has changed and will be denied access to RP's resources.

The key server generates a time sensitive key (TIK) that will specify the time a requestor is allowed to access the provider's resources. The server distributes TIK to Requestor RR_y and Provider RP_x. The server does not allocate resources. Hence the server load is typically low. Multiple servers may be deployed as servers are independent of each other to provide a truly distributed system.

Properties

- All messages are encrypted when communicating between server and nodes and between nodes.
- Neither the resource provider nor the resource requestor can dispute the time t_x^y the requestor is allowed to access the resource as the server sets the time and both the requestor and the provider are informed of this time.
- The provider cannot deny resources to the requestor as t_x^y is set by the server and delivered though k_t
- Each resource access session has a unique key to ensure that keys cannot be reused. This key can be used only for the current session for RR_y to access RP_x's resources. A different key will have to be generated for RR_y to access RP_x's resources in the future.
- At the latest, after the elapsed time, the provider will shut off resources to the requestor by changing the access code.
- If there is collusion to increase the participation factor of the free rider, the colluding resource provider will have to provide its resources to the free rider. This is how the game proceeds. Hence there is a cost to the provider and therefore it is not in the interest of the provider to provide resources to a free rider. Collusion involves malicious parties colluding to obtain resources of non-colluding honest parties. In this case, the loss is to one of the colluding parties, namely the provider. On the other hand, if the collusion results in a decrease of the participation factor, the free rider is adversely impacted. In this case, the loss is to the other colluding party. Hence, there is little incentive to collude.
- The limitations with this scheme include synchronization of the different clocks on the server, the requestor, and the provider.

Correctness property

if (tok_m, a_{RR}) is sent by the provider to the requestor and the Time Instant Key (TIK) k_t is output by the server on input $t_T = \lceil \lceil t_s \rceil + \lceil t_x^y \rceil + \lceil n \rceil \rceil$

then the requestor can access the provider's resources only if $t \in [t_0, t_T]$

else it returns a failure $\perp$.

Implementation

In a distributed cloud, we use Kademlia [17], a peer-to-peer distributed hash table for routing. We vision the implementation of the infrastructure required for a distributed cloud as an overlay network. In distributed cloud model [7], a distributed cloud computing and storage framework, node architecture, and resource discovery mechanism were implemented. In the current implementation we incorporated bargaining mechanisms that are built on top of this structured P2P overlay network. The proposed distributed cloud uses

Kademlia for nodes joining and leaving the system. In a distributed cloud after the resources are discovered, the user *RR* will use the bargain mechanism to select appropriate *RPs*. In Fig. 3 below, there are four providers. *RR* will initiate the bargaining process by sending an initial request message to all the *n RP*s discovered. Resource providers RP_i will then send their response to *RR*. Based on the game described in Sect. 5 above, the best RP_i will provide the resources for *RR*. To mitigate the free rider problem, we included an incentive-based mechanism for allocating resources, a participation-based strategy for identifying free riders, and a secure time-based key management system for limiting resource access.

The major communication required in our model is to perform resource discovery. We have used Kadelmia overlay along with the multi-valued hash table scheme to perform efficient resource discovery in Distributed cloud and our method provides O(log N) to find a resource. The other major communication is to connect to a trusted server which generates the TIK. This communication can be optimized and is minimal because the server location is known to all nodes. Computation load that incurs on nodes after the resources are selected is unknown and is mainly application dependent. It depends on what resource requestor tasks are what they want to perform on nodes selected. Trusted server calculates TIK and computation performed to generate TIK is very light and doesn't introduce any significant load.

Simulation

We simulated a distributed cloud computing environment using Kademlia. We used the King dataset [20] to simulate the latency between nodes. The King data set contains measurements of latencies between a set of DNS servers. This data set has been used to evaluate Vivaldi network coordinate system and other distributed systems. Each node has different attributes such as id, memory, capacity, memory, cores, and availability. Apart from maintaining routing table and a multi-valued hash table, each node also maintains a neighbor information table. We used Kademlia routing table in our simulator with each node maintaining 3 k-buckets with bucket size set to 1. A ping event will check the status and updates of node when it joins and allocates resources to other nodes. Each node updates its routing table, neighbor information table and multi-valued hash table when a new node joins and whenever there is a new node discovered. Apart from that each node updates its neighbor information table when a resource request has been satisfied. After a node has assigned its resource to another node it deducts the resources assigned. It then updates its multi valued hash table and then informs about the change to nodes in its routing table. Once the processing time is finished, a node updates its new availability information and multi valued hash table again.

The simulation network consisted of 10,000 nodes. Each node has a 32-bit unique ID generated randomly.

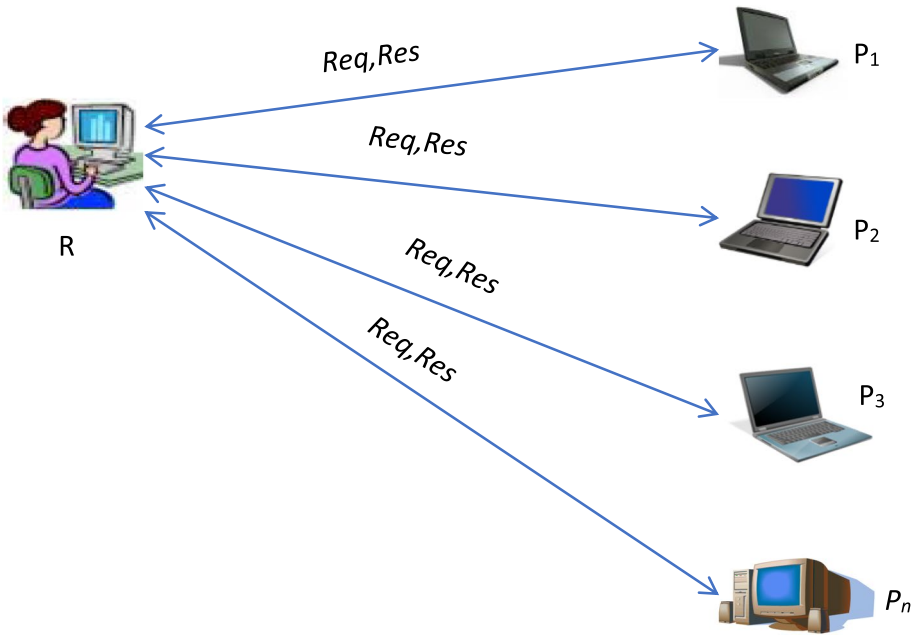

Fig. 3 Implementation of Distributed Cloud

Each node has mean memory of 8 GB; the minimum is 1 GB and maximum is 16 GB. Each node has cores which are randomly assigned form set of valid cores 1, 2,4,6,8 or 10. To make sure that simulations would be more realistic we have been biased towards lower core values when assigning them. This is the assumption we used because in the real world most of the machines offered would be small machines with basic computation power i.e., lower number of cores and low memory. So, we had 50% of nodes with 1 core, 25% of nodes with 2 cores, 10% of nodes with 4 cores, 8% of nodes with 6 cores, 4% of nodes with 8 cores, and 3% of nodes with 10 cores. We also made sure that most of the requests to the distributed cloud are more biased towards the lower values of cores and memory, i.e., users requesting nodes with lower computing power. We had 80% of nodes requesting 1 core and memory up to 4 GB, 10% requesting 2 cores and memory ranging 5 to 11 GB, 5% requesting 4 cores, 2% requesting 6 cores, 1% requesting 8 and 10 cores and all 4, 8 and 10 core machines requesting 12-16 GB memory.

Each node requests a resource or set of resources with randomly assigned capability. Requests are processed over a normal distribution with a time of 6 h; the minimum is 1 h, and the maximum is 6 h. We ran the experiments with 50% of nodes as free riders to see the effect of free riders on the system. We evaluated the system based on the number of searches required to find node or set of nodes and their success rates.

From Fig. 4 and Fig. 5, using the incentive-based model we propose, it is observed that with free riders in the system, not only does the number of required searches to find resources increase, but the success rate of finding the nodes decrease. This is because of the low participation factor. We see from Fig. 4 that the success rate of finding a resource is low and is reduced by almost 55%. From both these evaluations, we see that free riders in a system can decrease the stability of the system and penalizes genuine users as they may not be able to get the resources they want.

To validate our proposed participation factor scheme, we also ran the simulation to discover 10% and 50% of free riders in the system under three different scenarios. The scenarios used were, poor participation factor, medium participation factor and good participation factor. We calculated the time taken to find 10% and 50% free riders and are shown in Fig. 6. It is observed that the time taken to identify free riders when participation factor is poor is minimum. We also notice that it takes more time to identify free riders when participation factor is medium. When participation factor is medium, then users may be good to some users in providing resources and bad to other users. If there are users who are good to some users and bad with few others, it takes time to designate them as free riders by all the users. Finally, we see if participation factor is good, it takes less time to identify free riders than when participation factor is medium, but more time than when participation factor is poor. This is because a good participation factor means that nodes are given more time to access resources, this results in a greater number of iterations to update the participation factor ad identify a free rider. Because of internet speeds and network traffic, the timeframes may differ in real-world settings. However, the pattern will be identical to what we observed. Since the time taken to identify the free riders is very minimal, our proposed model is effective.

Conclusions

Existence of free riders in a network creates an implicit tension among resource providers as free riders do not provide resources to others. In the distributed cloud resource providers share resources in P2P fashion. Existence of free

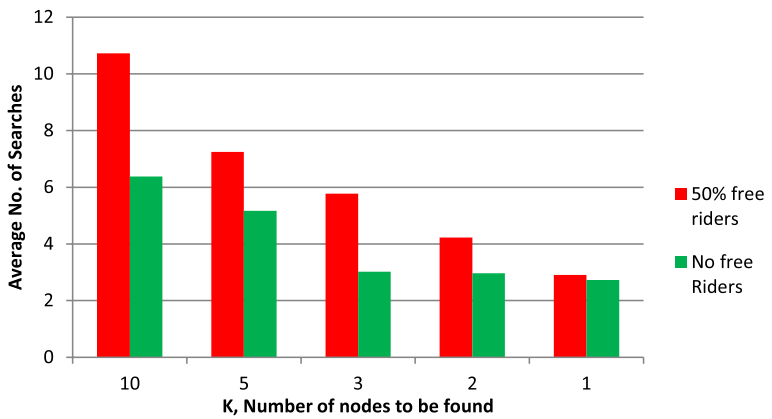

Fig. 4 Number of searches required to find k nodes

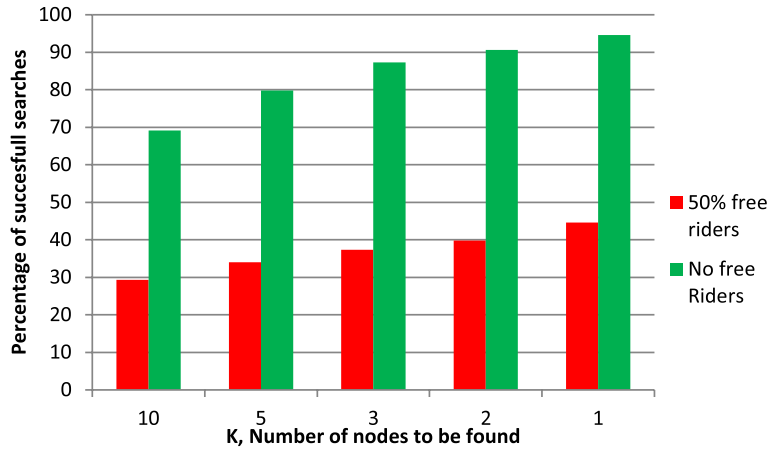

Fig. 5 Percentage of successful searches

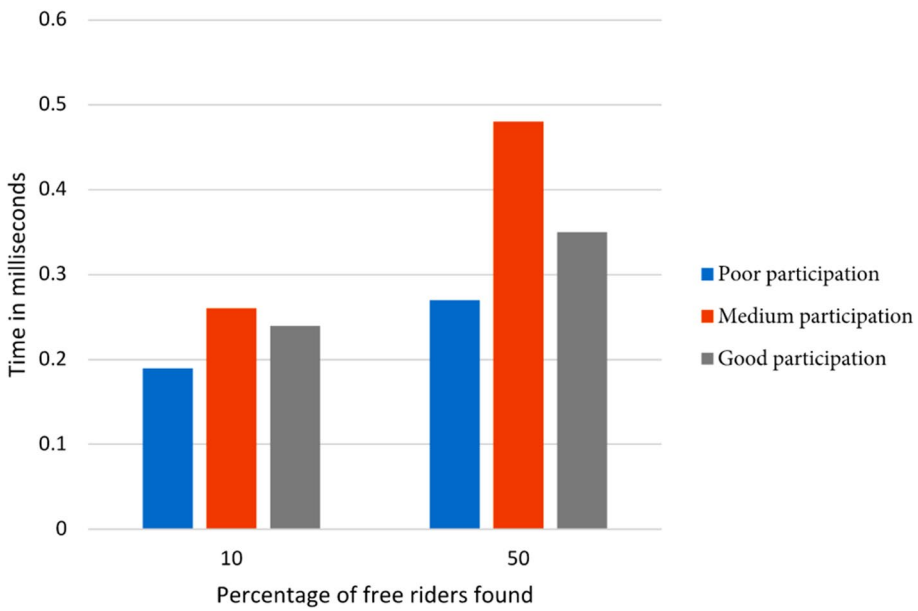

Fig. 6 Percentage of free riders found vs time taken

riders in distributed cloud will create many problems such as unwanted network traffic, system collapse and will discourage new users to join the network. In this paper, we propose a 3-pronged solution to mitigate free rider problem in distributed cloud environment. An incentive-based scheme is proposed to assign resources such as CPU time and memory based on whether the requestor of resources is a good user or a free rider. Our approach detects free riders using a participation scheme. A key management scheme based on a time instant key enables good users' sufficient time to access resources, whereas free riders are given limited or no time. Simulation results show that our scheme is effective detecting free riders and in limiting resource access to free riders. In this work we have limited the time–space domain of free riders. For future work we will look at other types of game theoretic approaches, derive models for resource provision, and measure performance overheads using our approach.

Acknowledgements
Part of this project has been supported by PSC-CUNY research award program.

Authors' contributions
Proposed method: Doyel Pal and Praveen Khethavath did all the simulation work and also some of the work on the game theory. Johnson Thomas did

some work on the game theory and proposed the key model. Utpal Mangala gave some valuable insights into the key model. He also provided helpful feedback on the final manuscript. All authors have read the final manuscript.

Funding
Part of this project has been supported by PSC-CUNY research award program.

Availability of data and materials
The supporting data used in this article for simulation is cited and are available from corresponding author upon request.

Declarations

Ethical approval and consent to participate
(should this be indicated as it is a health care based on Blockchain and Privacy Computing).
Not applicable

Consent for publication
(should this be indicated as it is a health care based on Blockchain and Privacy Computing).
Not applicable

Competing interests
All authors declare that they do not have competing interests.

Author details
[1]Mathematics, Engineering and Computer Science Department, LaGuardia Community College, CUNY, Long Island City, New York City, NY 11101, USA. [2]Department of Computer Science, Oklahoma State University, Stillwater, OK, USA. [3]IBM Global Services, Toronto, Canada.

Received: 6 September 2021 Accepted: 11 February 2022
Published online: 24 February 2022

References

1. Michael Armbrust, Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica and Matei Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing", Technical report EECS-2009–28, UC Berkeley, 2009
2. Peter Mell and Timothy Grance, "NIST definition of cloud computing, National Institute of Standards and Technology", Special Publication 800–145, 2011, "http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf", Retreived July 22, 2014
3. Buyya, R. and Chee Shin Yeo and Venugopal, S., "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities", Proccedings 10th IEEE International Conference on. High Performance Computing and Communications, HPCC'08. IEEE, 2008.
4. Amazon EC2, "http://aws.amazon.com/ec2/", Retrieved July 22, 2016
5. Cisco, "http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html"
6. Fox A (2011) Cloud computing-what's in it for me as a scientist. Science 331:406–407
7. Praveen Khethavath, Johnson P Thomas, and Eric Chan-Tin, "Towards an efficient Distributed cloud computing architecture", Peer-to-peer Networking and Applications (Springer), https://doi.org/10.1007/s12083-016-0468-x, 2016
8. Hughes, Daniel, Geoff Coulson, and James Walkerdine. "Free riding on Gnutella revisited: the bell tolls?" IEEE distributed systems online 6.6 (2005).
9. Ramaswamy, Lakshmish, and Ling Liu. "Free riding: A new challenge to peer-to-peer file sharing systems." System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on. IEEE, 2003.
10. Krishnan, Ramayya, et al. "The impact of free-riding on peer-to-peer networks." System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on. IEEE, 2004.
11. Karakaya M, Korpeoglu I, Ulusoy Ö (2009) Free riding in peer-to-peer networks. IEEE Internet Comput 13(2):92–98
12. Li X, Zhou F, Yang X (2011) A multi-dimensional trust evaluation model for large-scale P2P computing. Journal of Parallel and Distributed Computing 71(6):837–847
13. Feldman M, Chuang J (2005) Overcoming free-riding behavior in peer-to-peer systems. ACM sigecom exchanges 5(4):41–50
14. Wang Y, Cai Z, Yin G, Gao Y, Tong X, Han Q (2016) A game theory-based trust measurement model for social networks. Computational Social Networks 3(1):1–16
15. Tian C, Yang B (2011) Trust, a reputation and risk-based trust management framework for large-scale, fully decentralized overlay networks. Futur Gener Comput Syst 27(8):1135–1141
16. Boukerche A, Ren Y (2008) A trust-based security system for ubiquitous and pervasive computing environments. Comput Commun 31(18):4343–4351
17. P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In Proceedings of IPTPS02, Cambridge, USA, Mar. 2002.
18. M. J. Osborne and A. Rubenstein, A Course in Game Theory. MIT Press, 1994.
19. Pal, D., Khethavath, P., Thomas, J.P. and Chen, T., 2015, August. Multilevel Threshold Secret Sharing in Distributed Cloud. In International Symposium on Security in Computing and Communication (pp. 13–23). Springer International Publishing.
20. Index of /archive/p2psim/kingdata https://pdos.csail.mit.edu/archive/p2psim/kingdata/. 2016.
21. QinLiu, GuojunWang, JieWu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment" Information Sciences Volume 258, February 2014, Pages 355–370
22. Kenneth G. Paterson and Elizabeth A. Quaglia, "Time-Specic Encryption", International Conference on Security and Cryptography for Networks SCN 2010: Security and Cryptography for Networks pp 1–16

Publisher's Note