An Efficient and Trustworthy Resource Sharing Platform for Collaborative Cloud Computing

Haiying Shen*, *Senior Member, IEEE*, Guoxin Liu, *Student Member, IEEE*

Abstract—Advancements in cloud computing are leading to a promising future for collaborative cloud computing (CCC), where globally-scattered distributed cloud resources belonging to different organizations or individuals (i.e., entities) are collectively used in a cooperative manner to provide services. Due to the autonomous features of entities in CCC, the issues of resource management and reputation management must be jointly addressed in order to ensure the successful deployment of CCC. However, these two issues have typically been addressed separately in previous research efforts, and simply combining the two systems generates double overhead. Also, previous resource and reputation management methods are not sufficiently efficient or effective. By providing a single reputation value for each node, the methods cannot reflect the reputation of a node in providing individual types of resources. By always selecting the highest-reputed nodes, the methods fail to exploit node reputation in resource selection to fully and fairly utilize resources in the system and to meet users' diverse QoS demands. We propose a CCC platform, called Harmony, which integrates resource management and reputation management in a harmonious manner. Harmony incorporates three key innovations: integrated multi-faceted resource/reputation management, multi-QoS-oriented resource selection, and price-assisted resource/reputation control. The trace data we collected from an online trading platform implies the importance of multi-faceted reputation and the drawbacks of highest-reputed node selection. Simulations and trace-driven experiments on the real-world PlanetLab testbed show that Harmony outperforms existing resource management and reputation management systems in terms of QoS, efficiency and effectiveness.

Index Terms—Distributed systems, Reputation management, Resource management, Distributed hash tables, Cloud computing

1 Introduction

Cloud computing has become a popular computing paradigm, in which cloud providers offer scalable resources over the Internet to customers. Currently, many clouds, such as Amazon's EC2, Google's AppEngine, IBM's BlueCloud, and Microsoft's Azure, provide various services (e.g., storage and computing). For example, Amazon [1] (cloud provider) provides Dropbox [2] (cloud customer) the Simple Storage Service (S3) (cloud service). Cloud customers are charged by the actual usage of computing resources, storage, and bandwidth.

The demand for scalable resources in some applications has been increasing very rapidly. For example, Dropbox currently has 5 million users, three times the number last year. A single cloud may not be able to provide sufficient resources for an application (especially during a peak time). Also, researchers may need to build a virtual lab environment connecting multiple clouds for petascale supercomputing capabilities or for fully utilizing idle resources. Indeed, most desktop systems are underutilized in most organizations; they are idle around 95% of the time [3]. Thus, advancements in cloud computing are inevitably leading to a promising future for collaborative cloud computing (CCC), where globally-scattered distributed cloud resources belonging to different organizations or individuals (i.e., entities) are collectively pooled and used in a cooperative manner to provide services [4, 5].

As shown in Figure 1, a CCC platform interconnects physical resources to enable resource sharing between clouds, and provides a virtual view of a tremendous amount of resources to customers. This virtual organization is transparent to cloud customers. When a cloud does not have sufficient resources demanded by its customers, it finds and uses the resources in other clouds.

Importance of Resource and Reputation Management CCC operates in a large-scale environment involving thousands or millions of resources across disparate geographically distributed areas, and it is also inherently dynamic as entities may enter or leave the system and resource utilization and availability are continuously changing [4, 6]. This environment makes efficient resource management (resMgt) (i.e., resource location and resource utilization) a non-trivial task. Further, due to the autonomous and individual characteristics of entities in CCC, different nodes provide different quality of service (QoS) in resource provision. A node may provide low QoS because of system problems (e.g., machines break down due to insufficient cooling) or because it is not willing to provide high QoS in order to save costs. Also, nodes may be attacked by viruses and Trojan horse programs. This weakness is revealed in all the cloud platforms built by Google, IBM, and Amazon [7], and security has been recognized as an important factor in grids (the predecessor of clouds) [8]. Thus, resMgt needs reputation management (repMgt) to measure resource provision QoS for guiding resource provider selection [4, 7]. As in eBay and Amazon, a repMgt system computes each node's reputation value based on evaluations from others about its performance in order to provide guidance in selecting trustworthy resources.

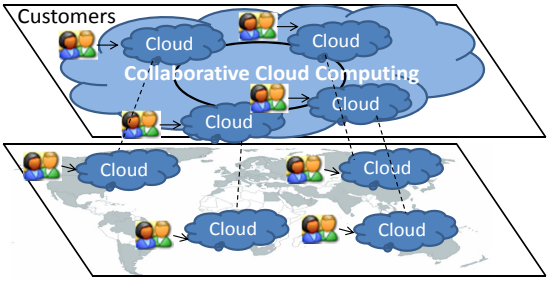


Fig. 1. An example of collaborative cloud computing.

To ensure the successful deployment of CCC, the issues of resMgt and repMgt must be jointly addressed for both efficient and trustworthy resource sharing in three tasks:

(1) Efficiently locating required trustworthy resources.
(2) Choosing resources from the located options.
(3) Fully utilizing the resources in the system while avoiding overloading any node.

Previous Methods and Challenges The three tasks must be executed in a distributed manner since centralized methods are not suitable for large-scale CCC. However, though many

* *Corresponding Author. Email: shenh@clemson.edu; Phone: (864) 656 5931; Fax: (864) 656 5910.*

The authors are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634. E-mail: {shenh, guoxinl}@clemson.edu

distributed resMgt and repMgt systems for grids have been proposed previously, and cloud resource orchestration (i.e., resource provision, configuration, utilization and decommission across a distributed set of physical resources in clouds) [6] has been studied in recent years, these two issues have typically been addressed separately. Simply building and combining individual resMgt and repMgt systems in CCC will generate doubled, prohibitively high overhead. Moreover, most previous resMgt and repMgt approaches are not sufficiently efficient or effective in the large-scale and dynamic environment of CCC.

Previous repMgt systems [9–12] neglect resource heterogeneity by assigning each node one reputation value for providing all of its resources. We claim that node reputation is multi-faceted and should be differentiated across multiple resources (e.g., CPU, bandwidth, and memory). For example, a person trusts a doctor for giving advice on medical issues but not on financial issues. Similarly, a node that performs well for computing services does not necessarily perform well for storage services. Thus, previous repMgt systems are not effective enough to provide correct guidance for trustworthy individual resource selection. In task (1), RepMgt needs to rely on resMgt for reputation differentiation across multiple resources.

Previous resMgt approaches only assume a single QoS demand of users, such as efficiency or security. Given a number of resource providers (i.e., servers), the efficiency-oriented resMgt policy would choose the one with the highest available resource, while the security-oriented repMgt policy would choose the one with the highest reputation. The former may lead to a low service success rate while the latter may overload the node with many resource requests. Thus, uncoordinated deployment of repMgt and resMgt will exhibit contradictory behaviors and significantly affect the effectiveness of both, finally leading to degraded overall performance. The results of the single-QoS-demand assumption and contradictory behaviors pose two challenges. First, in task (2), how can we jointly consider multiple QoS demands such as reputation, efficiency, and available resources in resource selection? Second, in task (3), how can we enable each node to actively control its reputation and resource supply so that it avoids being overloaded while gaining high reputation and profit?

Our Proposed Method By identifying and understanding the interdependencies between resMgt and repMgt, we introduce Harmony, a CCC platform with harmoniously integrated resMgt and repMgt. It can achieve enhanced and joint management of resources and reputation across distributed resources in CCC. Different from the previous resMgt and repMgt methods, Harmony enables a node to locate its desired resources and also find the reputation of the located resources, so that a client can choose resource providers not only by resource availability but also by the provider's reputation of providing the resource. In addition, Harmony can deal with the challenges of large scale and dynamism in the complex environment of CCC. The contributions of this work can be summarized as below:

(1) ***Preliminary study on real trace and experimental results.*** We analyzed the transaction and feedback rating data we collected from an online trading platform (Zol [13]). We found that some sellers have high QoS in providing some merchandise but offer low QoS in others, and buyers tend to buy merchandise from high-reputed sellers. The findings verify the importance of multi-faceted reputation and the drawback of the highest-reputed node selection policy.

(2) ***Integrated multi-faceted resource/reputation management.*** Relying on a distributed hash table overlay (DHT), Harmony offers multi-faceted reputation evaluation across multiple resources by indexing the resource information and the reputation of each type of resource to the same directory node. In this way, it enables nodes to simultaneously access the information and reputation of available individual resources.

(3) ***Multi-QoS-oriented resource selection.*** Unlike previous resMgt approaches that assume a single QoS demand of users, Harmony enables a client to perform resource selection with joint consideration of diverse QoS requirements, such as reputation, efficiency, distance, and price, with different priorities.

(4) ***Price-assisted resource/reputation control.*** In a *resource transaction*, a resource requester pays a resource provider (in the form of virtual credits) for its resource. The transactions are conducted in a distributed manner in Harmony. Harmony employs a trading model for resource transactions in resource sharing and leverages the resource price to control each node's resource use and reputation. It enables each node to adaptively adjust its resource price to maximize its profit and maintain a high reputation while avoiding being overloaded, in order to fully and fairly utilize resources in the system.

We have conducted extensive trace-driven experiments with PlanetLab [14] and simulations. Experimental results show the superior performance of Harmony in comparison with previous resMgt and repMgt systems, and the effectiveness of its three components. This work is the first to integrate repMgt with resMgt for multi-faceted node reputation evaluation to provide precise guidance for individual resource selection. In addition to CCC, Harmony can also be applied to other areas such as large-scale distributed systems, grids and P2P systems.

The preliminary work of this paper was presented in [15, 16]. We introduced an integrated resource and reputation management system in [15] and an efficient and locality-aware resource management system in [16]. In this paper, we combine the two systems and identified CCC as an example of large-scale distributed systems for the application area of this combined system. We propose a new system component called "price-assisted resource/reputation control." We also present new trace analysis results to prove the necessity of the proposed algorithms. We further extensively evaluate the performance of the system through many experiments.

The rest of this paper is structured as follows. Section 9 presents a concise review of representative resMgt, repMgt, and economic marketing approaches. Section 2 analyzes our crawled trace data and studies the relationship between resMgt and repMgt with experiments. Section 3 details the design of the three components of Harmony and Section 3.4 presents how the three components cooperate with each other in the Harmony system. Section 4 presents trace-driven experimental results in comparison with previous resMgt and repMgt approaches. Finally, Section 5 summarizes this paper with remarks on our future work. The supplemental file presents the pseudo-codes for two algorithms, additional PlanetLab experimental results, simulation results of Harmony in a large-scale system, and an overview of related work.

2 Preliminary Study

Figure 2 shows the three components of Harmony needed in each stage of resource marketing: location, selection and transaction. Below, we describe the motivations of the components.

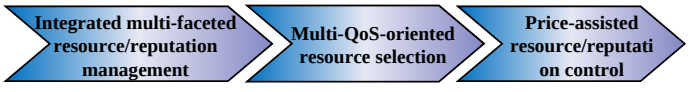


Fig. 2. Harmony components in resource market stages.

2.1 Motivation of Integrated Multi-faceted Res/Rep Mgt

We first study whether a high-reputed node (measured based on its resource provision services for all types of resources) provides high QoS for every type of resource. However, there are no resource markets with a reputation system. We thus studied an online merchandise trading platform, where

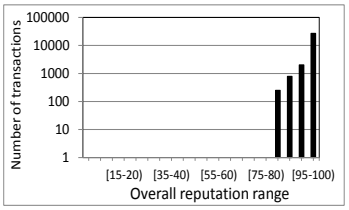
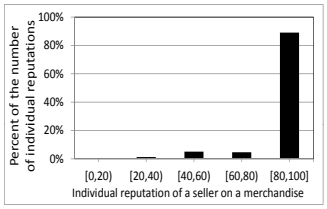
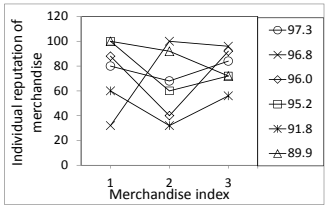


Fig. 3. Transactions.



(a) Individual reputation



(b) Individual reputations

Fig. 4. Importance of multi-faceted resource/reputation management.

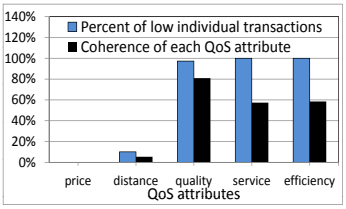


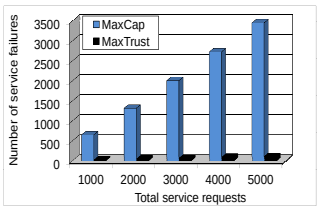
Fig. 5. QoS attributes.

each seller sells a variety of merchandise. The sellers and merchandise are regarded as nodes and resources, respectively. The data analytical results, to a certain extent, reflect the selling and buying behaviors.

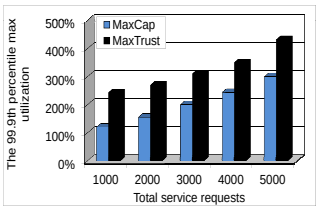
Zol is a top online trading platform in China similar to Amazon and eBay. Zol is chosen for market data analysis because neither Amazon nor eBay provides the historical rating record of each transaction. Zol provides the historical reputation record of each transaction, which enables to calculate the reputation for each type of a seller's merchandise for the multi-faceted reputation study. We collected trace data including 1,562,548 transaction records from Zol covering the period from 9/20/2006 to 6/26/2010. In addition to the overall reputation values of sellers, Zol provides the ratings within [0,100] for five QoS attributes for each transaction: 1) price, 2) distance, 3) quality, 4) service, and 5) efficiency. Figure 3 shows the number of transactions versus the seller's overall reputation from this entire trace dataset. The figure shows that clients tend to choose higher-reputed nodes for transactions. Thus, if the resource a node possesses is limited, the highest-reputed nodes can easily become overloaded.

In order to cluster the transactions by different types of merchandise, we identified all transactions by 60 merchandise keywords, and found 140,720 transactions of 50 sellers. The 60 merchandise are the first 60 merchandise of different brands in the Electronics category such as "Samsung mobile phone" and "Lenovo laptop". For these transactions, the transactions of the same type of merchandise of a seller are classified into a group. The average of the five ratings of each transaction is calculated as the seller's QoS for this transaction. Then, the average QoS of each group is calculated as the reputation of the seller for this type of merchandise (named as *individual reputation*). Finally, the individual reputation on each merchandise type of each seller is derived. The results show that the lowest overall reputation of these sellers is 89, which means that these sellers are relatively high-reputed. Figure 4(a) shows the distribution of the individual reputations of these 50 sellers. It shows that 90% of the individual reputations of these sellers are in the range of [80,100], 4.4% are in [60,80], and 5.1% are in [40,60]. From the identified 50 sellers, 13 sellers who have 3 types of merchandise in common are identified. Figure 4(b) plots the individual reputation values for the 3 types of merchandise of the six top overall reputed sellers. The values on the right side of the figure are the sellers' overall reputations. It shows that the individual reputation for each type of a seller's merchandise fluctuates greatly, even dropping below 30 sometimes. The results from the two figures imply that even high-reputed sellers sometimes offer low QoS for certain types of merchandise, which supports our claim that resource reputation should be multi-faceted. When choosing a resource supplier, the individual reputation of the merchandise type should be referred to rather than the overall reputation of suppliers.

In the 140,720 transactions of the 50 sellers, the rating of a QoS attribute that is no more than 40 is considered to be a low rating; similarly an individual reputation that is no more than 40 is considered to be a low reputation. Figure 5 shows the percentage of transactions with low ratings in each QoS



(a) Failures in services



(b) Node utilization

Fig. 6. Importance of joint efficiency and trust consideration.

attribute out of all low individual reputation transactions. It shows that 100% have low ratings in service and in efficiency, 97% have low ratings in distance, and 10% have low ratings in quality. This indicates that 97% of transactions of merchandise with low individual reputations receive low ratings in the distance, service and efficiency QoS attributes. Thus, there is no QoS attribute whose extremely low rating leads to a low individual reputation for a merchandise type. Also, a low individual reputation merchandise may have a high rating in a QoS attribute such as price as shown in the figure.

N_a is used to denote the set of transactions of low ratings in a QoS attribute a, and N_t to denote the set of transactions of low individual reputation. The coherence value of QoS attribute a is defined as $|N_t \cap N_a|/|N_t \cup N_a - N_t \cap N_a|$, which is the correlation between QoS attribute a and the individual reputation. A higher coherence value for a QoS attribute means that a low rating of this QoS attribute is more likely to lead to low individual reputation. Figure 5 also shows the relationship between each QoS attribute and individual reputation. It shows that the quality, service, and efficiency QoS attributes have high coherence values and hence are the primary factors in the low rating of the individual reputation; however, Figure 5 shows that most transactions with low individual reputation may still have relatively high ratings in the price and distance QoS attributes. Therefore, a seller's individual reputation cannot reflect its QoS for each QoS attribute, which confirms the need to consider multiple QoS attributes in selecting resources. Based on this observation, Section 3.2 introduces a multi-QoS-oriented resource selection algorithm that enables clients to choose resources based on their priority considerations of the different QoS attributes.

2.2 Motivation of Multi-QoS-oriented Resource Selection and Price-assisted Control

Simply combining multi-resMgt and repMgt will lead to a few problems. First, resMgt and repMgt always have their own infrastructures. For example, Mercury [17] partitions nodes into groups based on resources for resMgt, and PowerTrust [11] builds a P2P system with links connecting interacting nodes for repMgt. Maintaining two infrastructures generates high maintenance overhead. Second, most resMgt approaches are driven by either efficiency or security through choosing the highest-reputed or highest-capacity node. Hence, a direct combination will lead to contradictory behaviors. Third, since a node refers to the overall reputation in selecting individual resources, it may receive incorrect guidance because a high-overall-reputed node may provide low QoS for individual resources.

With the assumption that there is a single resource bottleneck, an experiment on two node selection policies was conducted: `MaxTrust` [18] and `MaxCap` [17, 19, 20], each of which chooses the node with the highest reputation and highest available capacity (i.e., resource), respectively. The simulation assumed three types of nodes: altruistic, neutral and egotistic, each of which provides its service successfully with a probability 1, 0.5, and 0.1, respectively. Each node is randomly assigned to one of the three types, and every request has 10 servers that are able to satisfy the request. The *utilization* of a node is defined as the ratio of its load to its capacity. The maximum utilization value of each node during the experiment was recorded. The 99.9th percentile value in this group of maximum utilization values is referred to as *the 99.9th percentile maximum utilization*. A *service failure* occurs when the selected node is unwilling to provide a service. Figures 6(a) and (b) show the number of service failures and the 99.9th percentile maximum utilization of different policies. It shows that the efficiency-oriented policy, `MaxCap`, performs best in controlling node utilization but incurs a high service failure rate; in contrast, the trust-oriented policy, `MaxTrust`, performs the best in controlling service failure rate but leads to high node utilization. The fundamental reason for these severe problems is the neglect of the interdependencies between repMgt and resMgt; the uncoordinated deployment of either one will exhibit contradictory behaviors and significantly reduce the effectiveness of both. Therefore, a method is needed to jointly consider efficiency and trust in resource selection, and also need a method to avoid overloading nodes with resource requests.

3 The Design of Harmony

3.1 Integrated Multi-Faceted Res/Rep Management

Overview of Cycloid Harmony leverages the Cycloid hierarchically structured P2P overlay [21] for its substrate. It has at most $n{=}d \cdot 2^d$ nodes, where d is its dimension. The upper layer in Figure 7 shows an example of a Cycloid with $d = 3$. Each Cycloid node's ID is represented by a pair of indices $(k, a_{d-1}a_{d-2}\dots a_0)$, where $k \in [0, d-1]$ is a cyclic index and $a_{d-1}a_{d-2}.....a_0 \in [0, 2^d - 1]$ is a cubical index. For a given key or node IP address, its cyclic index is its consistent hash value [22] modulated by d and its cubical index is the hash value divided by d. All nodes are grouped into different clusters, which are identified by $a_{d-1}a_{d-2}.....a_0$. Within a cluster, the nodes are differentiated by k. In the Cycloid key assignment policy, an object/key is assigned to the node whose ID is closest to the object/key's ID. It provides two main functions: $Insert(ID, object)$ and $Lookup(ID)$ to store an object in its owner node and to retrieve the object, respectively. It achieves a time complexity of $O(\log n)$ per lookup request by using a constant 7 neighbors per node.

Our previous work [23] introduced the *Hilbert number* ($\mathcal{H}$), which represents a node's geographical proximity so that the distances between nodes can be inferred from the closeness of their $\mathcal{H}$ values. Physically close nodes have closer $\mathcal{H}$s. As shown in Figure 7, Harmony builds all nodes in CCC into a locality-aware Cycloid [24], where physically close nodes constitute a cluster and a node's neighbors are physically close to itself. To achieve this, Harmony uses a node's $\mathcal{H}$ as its cubical index. When a node chooses a neighbor for each entry in its neighbor table, it chooses the proximity-closest one from the options whose IDs satisfy the requirement of the entry.

Overview of Harmony We assume that resource types (e.g., CPU, bandwidth and memory) are globally defined and known by every node. The resource information (denoted by I_r) includes the resource provider's IP address, resource type, available amount, resource physical location, price, etc. A

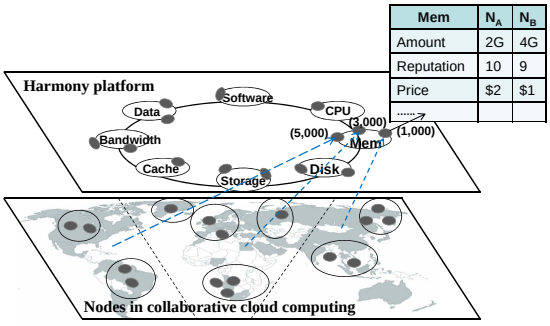


Fig. 7. The Harmony platform.

general distributed method for resource location is to store resource availability information in some directory nodes, and forward the resource requests to the corresponding directory nodes [17, 19, 20, 25–28]. Similarly, a general distributed method for repMgt is to store reputation information of nodes in some directory nodes, and forward the reputation requests to the corresponding directory nodes [9–12]. As in the previous repMgt systems, we assume that the directory nodes are trustable, and we leave the study of untrustworthy directory nodes as our future work. In the reputation system, nodes may dishonestly report the reputation feedback of their received service. We can use the previous works [10] that aim to more accurately evaluate node reputation to handle this problem. As the focus of this work is how to collect reputation feedbacks, based on which the node reputation is calculated. The techniques for accurate reputation calculation can be directly adopted by this work and they are orthogonal to our study in this paper. For multi-faceted resource/reputation management, I_r should be differentiated based on resource type, the reputation of a node should be differentiated based on the QoS of providing different types of resources (denoted by R_r), and a node's I_r and R_r should be stored in the same directory node. This enables a resource requester to find its requested resources (e.g., "CPU speed=1000MHz" and "Memory=1024MB") along with the reputations of resource providers in providing the requested resources, which are used for selecting providers.

To achieve this goal, as shown in the upper layer in Figure 7, Harmony pools all the resource information of each resource type into a cluster. It also distinguishes the reputation feedbacks for a resource provider by resource types, and stores the feedback into the corresponding resource cluster. For example, the reputation feedback of a node's "Mem" resource provision is mapped to the cluster for the "Mem" resource. As a result, for each specific resource type, Harmony pools together the I_r of available resources and R_r of resource providers in providing this resource.

Within each cluster, Harmony further groups the resource information of physically close nodes into one node in order to enable requesters to find physically close resources. Since all resource information I_r of each resource type in the system is stored in a cluster, to search for one resource, a directory node only needs to probe nodes in its cluster rather than executing system-wide probing. Pooling together the information of the same resource into a smaller cluster of physically close nodes reduces the probing latency and cost, thus improving resource discovery efficiency.

Details of Harmony With the design described above, each node periodically reports its available resources to directory nodes and sends resource requests to directory nodes when it needs resources. The directory nodes collect I_r and requests, and function as matchmakers between resource requesters and providers. The pseudocode of these three procedures is shown in Algorithm 1 in the supplementary file. Below, we present the details of each procedure.

As the *STORE* procedure in Algorithm 1 shows, for resource reporting, node i reports its available resource r by $Insert((\mathcal{H}_i, H_r), I_r)$, where H_r is the consistent value of resource r's name. Based on the Cycloid key assignment policy, the I_r is stored in the node whose ID is closest to $(\mathcal{H}_i, H_r)$. Recall that in a Cycloid ID, the second index differentiates the clusters and the first index differentiates the nodes within one cluster. Then, the I_r with the same H_r is stored in the same cluster, and $\widehat{I_r} \in I_r$ with the same $\mathcal{H}_i$ is stored in the same node in the cluster. Consequently, the resource information for the same resource type is collected in the same cluster, and the information is further distributed among the nodes within the cluster based on resource proximity.

In Figure 7, for example, the I_r of the memory resource of physically close nodes is forwarded to the same node in the "Mem" cluster. As the *REQUEST* procedure in Algorithm 1 shows, to query multiple resources, node i sends out $Lookup(\mathcal{H}_i, H_r)$ requests along with its desired amount, time period, reputation, price, and so on, with one request for one resource type. Based on the Cycloid routing algorithm, the request is forwarded to the owner node of $(\mathcal{H}_i, H_r)$ (i.e., the requested resource's directory node), which stores the information of the requested resource that is physically close to the requester. As the *PROCESS* procedure in Algorithm 1 shows, if the directory node has no I_r satisfying the requests (i.e., the amount and reputation are no less than the requested values and the price is no higher than the requested value), it probes its neighbor nodes in its cluster. After the directory node finds I_r satisfying the request, it uses the *multi-QoS-oriented resource selection* algorithm (Section 3.2) to select the best server(s) for the client, and then replies to the client with the results. Since all I_r of the requested resource in the system is in the cluster, if there's no I_r satisfying the request in the cluster, then there is no satisfying resource in the system. The directory node then sends a search failure response to the requester.

Below, we explain how a directory node probes its neighbor nodes in its cluster with the consideration of both locality and dynamism. Since the $\mathcal{H}$ represents the physical closeness of nodes, the I_r of physically close nodes gathers in the same or logically close nodes. Hence, with locality consideration, a node should probe its logically close neighbors in order to map physically close resource requesters and providers. Specifically, a directory node's request is forwarded sequentially along the nodes in its cluster. However, such sequential probing may fail due to node dynamism. It is known that randomized probing is an effective way to handle dynamism. It was also proved in [29] that 2-way randomized probing is effective in dealing with dynamism and could achieve faster speed over 1-way probing, but $m(>2)$-way probing may not result in much additional improvement. Therefore, Harmony uses a 2-way randomized probing method. Specifically, the node randomly generates two cyclic IDs within an increasing range of proximity in two directions, and probes the nodes with the random IDs. For example, if a node's ID is (5,200), it randomly generates two cyclic IDs within 2 and 8 given a range of 3. Suppose the generated numbers are 6 and 3. The node then probes nodes (6,200) and (3,200) at the same time. If the requested resource still is not found, the node increases the proximity range and repeats the same process. This method introduces a factor of randomness in the probing process in a range of proximity to deal with dynamism and locality simultaneously. Probing in an increasing range of proximity helps to map physically close resource requesters and providers, improving resource management efficiency.

Upon receiving the satisfying I_r, the client further chooses the servers with a highest reputation and lowest resource price (Section 3.3), and then randomly chooses one to ask for the resource. After the client finishes using the requested resource, it reports its reputation feedback on the server's resource provision service using $Insert((\mathcal{H}_s, H_r), R_r)$, where s represents the server. According to the Cycloid key assignment policy, the feedback receiver is exactly the directory node for resource r of server s. Therefore, all R_r of server s in providing this resource, say "Mem", are collected in this directory node. The directory node periodically calculates the reputation value of node s in providing the "Mem" resource based on the feedback using a reputation calculation method [9–12]. Recall that all I_r of "Mem" of server s is also reported to this directory node. Thus, in Harmony, for a specific resource, the I_r of a resource and R_r of the nodes offering this resource are stored in the same directory node. Node i can use $Lookup(\mathcal{H}_i, H_r)$ to query a resource and the reputations of providers offering that specific resource.

P2P's self-organization mechanism helps Harmony to handle node dynamism. In this mechanism, nodes update their neighbors periodically and transfer resource and reputation information based on the DHT key assignment policy when joining or leaving. Also, before a node departs from the system or after a node joins in the system, it notifies the directory nodes that store I_r of its resources. Thus, a node's I_r is always stored in its directory nodes even in dynamism, and the $Lookup(ID)$ requests can always be forwarded to the directory nodes. In a very large-scale distributed system, directory nodes may become bottlenecks. Our experimental results in Section 8.2 show that Harmony achieves a more balanced load distribution among directory nodes than the previous DHT-based resource management systems due to its two-layer hierarchical resource information distribution. When a directory node is overloaded, it can use the load balancing algorithm introduced in [29] to move its load to a lightly loaded node and maintain an index to the node. Harmony's resource/reputation management also features three characteristics: (1) locality-awareness and dynamism-resilience; (2) low maintenance overhead by relying on a single DHT for both resource and reputation management; and (3) multi-resource management using one DHT.

3.2 Multi-QoS-oriented Resource Selection

After a directory node locates the resource providers that have the required reputation, available amount, and price, it needs to choose provider(s) for the requester. The final QoS offered by a provider is determined by a number of factors such as efficiency, trustworthiness, distance, security and price. We call these factors QoS demands (or attributes). When choosing from a number of providers, most previous approaches rigidly consider a single QoS demand at a time. However, different tasks have different requirements. For time-critical tasks, distance should be given priority. For a large computing task, efficiency should be the main deciding factor. Further, a server's distances to different clients are different. This means a server's final QoS for client i does not necessarily represent its QoS for client j. Also, a task or user may have multiple demands with different priorities. A challenge here is how to consider individual or combined QoS attributes, and a user's desired priorities of the attributes in provider selection.

Harmony solves this problem by unifying all attribute values and a client's considered attribute priority into an overall QoS metric. eBay's reputation system asks users to provide feedback on different aspects such as item description and shipping charge. Similarly, Harmony utilizes a list of QoS attributes. It requires nodes to give ratings for each QoS attribute and overall QoS for a resource service in addition to the reputation for a server. As the reputation feedback, the QoS ratings are also collected at the directory node of the provided resource of the server. The overall QoS is actually a result of the combined influence from the QoS attributes. However, it is not easy to

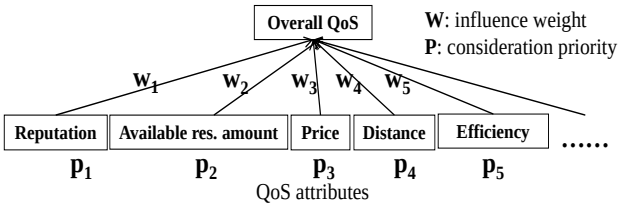


Fig. 8. A neural network model for resource selection.

detect how the different attributes influence the overall QoS. Harmony depends on a neural network [30] (e.g., Bayesian network [31]) to find out the influence weight of each attribute on the overall QoS value, and further considers users' attribute consideration priority. A neural network can be used to derive meaning from complicated or imprecise data, and extract patterns and detect trends. It uses adaptive learning to learn how to do tasks based on the data given for training, and then functions as an "expert" on the data it has been given to analyze.

In Harmony, each directory node builds a neural network model as shown in Figure 8. Its inputs are the ratings of each QoS attribute, denoted by $\{A_1, A_2, \cdots, A_5, \cdots\}$, and its output is the overall QoS value for a provider. The training process of the neural network is the process of determining the weight of influence of each attribute on the QoS, denoted by $\{w_1, w_2, \cdots, w_5, \cdots\}$. These weights reflect the normal degree of influence from different QoS attributes on the overall QoS calculated by the collected QoS ratings from many nodes. Thus, the activation function is $QoS = \sum_{i=1}^{n} w_i * A_i$. After building the neural network, each directory node trains its neural network model periodically using its newly collected QoS ratings to keep the model and the calculated weights w_i up-to-date. A requester sends its consideration priority weight $p_i(\sum p_i = 1)$ along with its resource request. After the directory node locates satisfying resource providers, it calculates the overall QoS value for each server option by considering both the normal influence of QoS attributes on the overall QoS and the requester's consideration priority. To do this, the directory node temporarily changes w in the neural network model according to customer c's requests: $w_{i,c} = p_{i,c}w_i$, as shown in Figure 8. It then inputs each server's attribute values $\{A_1, A_2, \cdots, A_5, \cdots\}$ into the neural network. The output of the neural network is the overall QoS. Finally, the directory node determines the server(s) with the highest overall QoS value. Thus, Harmony jointly takes into account the client's considered priority on different attributes and the influence weights of attributes on the final rating in server selection.

3.3 Price-assisted Resource/Reputation Control

In managing decentralized resources, Harmony employs a resource trading model [32–35], which is recognized as an effective way to provide incentives for nodes to provide high QoS and thwart uncooperative behaviors. In the model, a node pays credits to a resource provider for offered resources, and a resource provider specifies the price of its resources. The credits could be either virtual money or real money. The price is the amount of credits to use one unit of resource for one time unit. Consequently, in order to use others' resources, a node must provide its resources to others (or pay real money).

We take one resource type as an example to describe the price-assisted resource/reputation control scheme. A provider normally specifies the price of its resources according to the reputation value, load, and the desirability of the resources. Resources with higher reputations, lower loads, and higher demand (frequently requested) should have high prices. Therefore, in order to earn more income, each node is motivated to provide high QoS to maintain high reputation, while avoiding being overloaded.

Previous repMgt methods always encourage nodes to choose the highest-reputed node as the server. However, with the highest-reputed server selection policy, a high-reputed server easily becomes overloaded. This server selection policy is effective when nodes have unlimited resources, but it does not hold true in CCC, where each node has limited resources. Always choosing the highest-reputed node will overload that node, which then cannot offer high-QoS and receives low ratings from others. Then, the node will have low reputation and few chances to be chosen as a server in order to earn credits and increase its reputation. Though a node in Harmony chooses a server with a record of sufficient resources in the directory node, the server still would be overloaded due to delayed updates of the resource information or many simultaneous requests.

Therefore, a challenge here is how to enable a node to keep a high reputation, fully utilize its resources and avoid being overloaded. We notice that among the different QoS attributes in Figure 8, price is the only attribute that can be controlled by a provider. By fine-tuning its price, a node can control the calculated overall QoS, thus controlling its own load and reputation. Hence, Harmony takes advantage of the price to avoid overloading nodes and strengthen cooperative and high-QoS resource sharing. Specifically, a node adaptively adjusts its resource price according to its load in order to always remain high-reputed and avoid being overloaded while gaining the highest income.

We define a load factor $f = l/c$, where l is the amount of a resource a node has provided to others, and c is the total amount of that resource the node owns. When a node's $f > 1$, it is overloaded. A node periodically checks its f. If the node's $f > \alpha$ $(0.8 \leq \alpha < 1)$, it increases its price by one price unit to discourage requesters and avoid being overloaded. Otherwise, it decreases its price by one price unit in order to promote its resource usage to raise its own reputation and income. We choose $\alpha < 1$ rather than $\alpha = 1$ in order to avoid delayed responses to the node's overloaded status. This control method is used to deal with the overloaded and underloaded resource utilization situations. Based on economic theory [36], as long as the resource demand and supply are stable, the resultant equilibrium price will be stable.

The price-assisted resource/reputation control scheme prevents uncooperative behaviors, encourages nodes to provide high QoS, and allows nodes to adaptively adjust their load to offer high QoS. As a result, all resources in the system are fully and fairly utilized, nodes are not overloaded, and a node's reputation can truly reflect its QoS in offering resources without the influence of the overloaded status. To address the problem of low reputation for newly joined nodes, Harmony assigns the nodes a certain amount of starting virtual credits that can be used for building initial reputation.

3.4 Combination of Three Components

Algorithm 2 in the supplementary file shows the pseudocode of the algorithm executed by node i in Harmony combining three components. In the *integrated multi-faceted resource/reputation management*, a node periodically reports its specified resource prices along with its available resource amount. When a node needs a resource, it sends a request with its desired price, amount, time period, and provider's reputation. After receiving a request, a directory node searches its directory, and finds all providers that have qualified resources satisfying the requester's requirements. It then uses the *multi-QoS-oriented node selection algorithm* to identify the resource providers with the highest overall QoS values, and notifies the requester of the providers. The requester then asks its selected providers for resources. Based on the *price-assisted resource/reputation control*, the resource receiver pays

a resource provider for the provided resources. Also, nodes periodically check their load and adjust their resource prices accordingly, in order to remain highly reputed and maximize their profits, while avoiding being overloaded. If a node's reputation for a resource is increased, it has a higher probability of being selected as a resource provider. Then, it gains more opportunities to earn credits and can have enough credits to buy its desired resources. On the other hand, if a node's reputation for a resource is decreased, it has a lower probability of being selected as a resource provider. Then, it has fewer opportunities to earn credits and may not have enough credits to buy its desired resources. As a result, nodes are motivated to increase their reputation by providing high QoS. The time period for the neural network model training and load factor calculation should be determined with consideration of several factors, including the performance requirement, the frequency of transactions, the overhead in the system, etc. A shorter time period leads to higher effectiveness in QoS predication accuracy and load controlling but generates higher overhead. We leave this optimization as our future work.

4 Performance Evaluation on PlanetLab

To validate the design of Harmony, we implemented a prototype on PlanetLab and conducted trace-driven experiments. Since we can stably access around 234 nodes in PlanetLab, we set the number of nodes in the system to 234. We created 12 resource types in our system. In order to derive node overall reputations and individual reputations for these resource types, we first identified 13 sellers from the Zol trace data with 3 merchandise types in common. We then mapped this trace data to the 234 nodes and 12 merchandise types. We normalized the reputation values from [0,100] to [0,10]. The lowest overall reputation of these sellers is 8.9 in the trace data. To simulate an environment with high-, medium- and low-reputed nodes, we generated synthetic data for 1/3 nodes with overall and individual reputations randomly chosen from [1,3] and [4,6], respectively. Their individual reputations were set equal to their overall reputations. On PlanetLab, we selected 21 nodes (7 from the US, Europe and Asia, respectively) as landmark nodes for calculating node Hilbert numbers. We randomly chose 8 nodes as requesters in America, Europe, and Asia, respectively. In the experiments, unless otherwise specified, each requester sends one request every 10s for a resource randomly chosen from the 12 types. A different request initiation rate would not change the relative performance differences between the systems under evaluation. A resource provider with individual reputation t for a resource has $t/10$ probability to provide this resource. We report the 10 average values of each group of 500 requests over time as the experimental results.

In order to show the importance of integrating resource management and reputation management, we first evaluated Harmony in comparison with Harmony without reputation management (denoted by resMgt) and the PowerTrust reputation management system [11]. To make the methods comparable, we use Harmony's structure and resource discovery algorithms for PowerTrust. These methods are only different in resource selection. After locating resource providers, Harmony chooses a lightly loaded provider with the highest individual reputation, PowerTrust chooses the provider with the highest overall reputation, and resMgt randomly chooses a provider among lightly loaded nodes.

We compared Harmony with a Single-DHT method [25] and a Multi-DHT method [20] in order to show Harmony's higher efficiency in the complex environment of large-scale and dynamic CCC. Single-DHT relies on a single DHT, in which a resource ID owner is the directory node for all information of this resource. Multi-DHT relies on multiple DHTs, in which one DHT is responsible for one resource type, and resource information is distributed among all DHT nodes based on their resource values.

4.1 Integrated Multi-Faceted Res/Rep Management

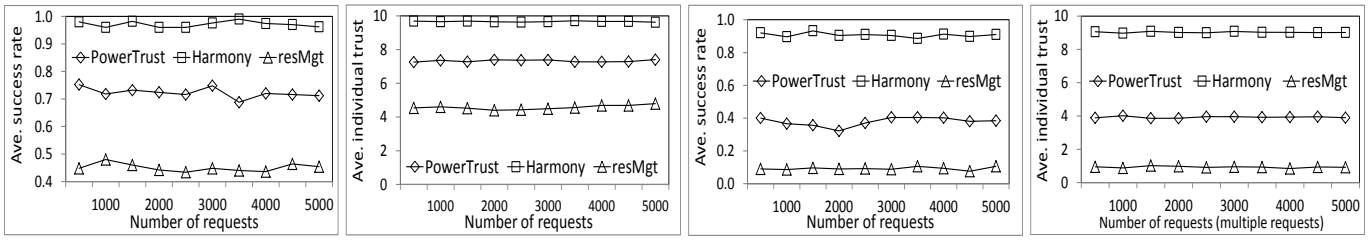
Resource management needs reputation management to provide a cooperative environment for resource sharing. Otherwise, a node cannot know which resources are trustworthy. Resource management in turn facilitates reputation management to evaluate multi-faceted node reputation in providing different resources. Therefore, resMgt may choose nodes unwilling to provide services and generate many service failures. PowerTrust may choose overloaded nodes that cannot process requests successfully. By integrating both resource management and reputation management, Harmony enables joint consideration of node reputation and load status and chooses lightly loaded high-reputed nodes that can always process requests successfully. Below, we present experimental results to show the importance of integrating resource management and reputation management.

4.1.1 Trustworthy Resource Sharing

We first tested different methods when all requests are single-resource requests. In order to see the effect of reputation management alone, we assumed that nodes do not drop requests when overloaded but queue the requests for processing later on. Figure 9(a) shows the average success rate of each system, which is measured by the ratio of successfully resolved resource requests over total requests. A request is successfully resolved if the selected server has provided its requested resource. We see that Harmony achieves a success rate of over 96%, while PowerTrust achieves around 73% and resMgt achieves around 44%. ResMgt selects a resource without considering reputation and may choose a resource provider with low reputation for the requested resource, leading to a low success rate. PowerTrust always selects the highest-overall-reputed provider. As verified by the trace, a node with a high overall reputation may provide low QoS for another resource due to either unwillingness or overloaded status. Therefore, Harmony significantly outperforms PowerTrust by always selecting the supplier with the highest individual, rather than overall, reputation. Figure 9(b) shows the average individual reputation of every group of 500 selected resource providers for the requested resources, which follows Harmony>PowerTrust>resMgt. The experimental results confirm the effectiveness of multi-faceted reputation management and its importance in guiding trustworthy resource selection.

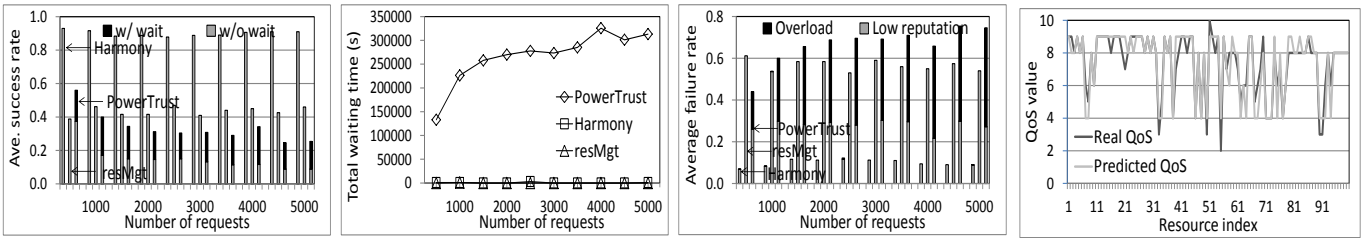
We then tested different methods when all requests are multiple-resource requests. Figure 9(c) shows the average success rate of each method, with each request calling for three resources. A multi-resource request is successfully resolved only after all three resources are successfully discovered. The result shows that Harmony>PowerTrust>resMgt due to the same reasons as in Figure 9(a) and Figure 9(b). Comparing Figure 9(c) with Figure 9(a), we observe that the average success rate of Harmony decreases about 0.06, while those of PowerTrust and resMgt decrease around 0.34 and 0.35, respectively. This is because if one of the three resource suppliers has a low individual reputation, the final request failure is low. We also find that Harmony outperforms PowerTrust and resMgt more significantly for multiple-resource requests because it can ensure the success rate of each of the three selected suppliers by considering multi-faceted reputations for different resources.

We used the product of the individual reputations of the selected three resource providers of a request as the individual reputation result of this request. Figure 9(d) shows the average individual reputation for each group of 500 multi-resource requests over time. The experimental result also follows



(a) Ave. success rate (single-res.) (b) Average individual reputation (single-res.) (c) Ave. success rate (multiple-res.) (d) Average individual reputation (multiple-res.)

Fig. 9. Trustworthy resource sharing.



(a) Ave. success rate (b) Total waiting time (c) Ave. failure rate

Fig. 10. Efficient resource sharing.

Fig. 11. Predicted and real QoS values.

Harmony>PowerTrust>resMgt, which is consistent with the success rate result in Figure 9(c). Also, the result of each method in Figure 9(d) is lower than that in Figure 9(b). This is caused by the same reasons for the differences between Figure 9(a) and Figure 9(c), which confirms the importance of considering individual reputation in selecting resource providers, especially for multi-resource requests.

4.1.2 Efficient Resource Sharing

Without resource management, reputation management cannot tell if a resource supplier has sufficient available resources. We then tested the importance of resource management to reputation management. In this experiment, each node maintains a waiting queue; an overloaded resource provider inserts its received request into its waiting queue. The requests staying in the queue for more than 100s are dropped. Therefore, a resource provision failure is caused by either a provider's overloaded status or its unwillingness to provide a service as reflected by its reputation. The Pareto distribution reflects the real world in which the amounts of available resources vary by different orders of magnitude [29, 37–39]. Thus, we used a Pareto distribution in determining a node's capacity for a resource type, a request's requested amount, and time period. We set the shape parameter to 2, and the scale parameters to 100, 40, and 200 for the above three parameters, respectively. We arbitrarily set the values in their reasonable ranges. Different setting values will not change the relative performance differences of the methods.

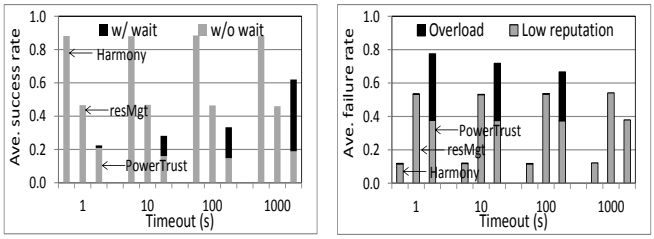
Figure 10(a) shows the average success rates for groups of 500 requests by each method. The black color represents delayed successful requests that have waited in the queue before being processed, and the grey color represents successful requests with no delay. We see that PowerTrust generates a large number of delayed successful requests, while the other methods generate no delayed requests. Figure 10(b) shows the total waiting time for each group of 500 requests, including failed requests. We see that PowerTrust generates high delay for a request, while Harmony and resMgt produce little or no delay, which is consistent with Figure 10(a). This is because PowerTrust always chooses the highest-overall-reputed nodes as resource providers without considering node load. These nodes receive too many requests, causing many to wait in the queues. Since both Harmony and resMgt select lightly loaded nodes as resource providers, they generate few delayed requests. Figure 10(a) also shows that the success rates in most cases follow Harmony>resMgt>PowerTrust. In order to further explore the trend, we show in Figure 10(c) the failure rate due to provider overload or unwillingness to serve for groups of 500 requests for each method. We see that PowerTrust generates a higher failure rate due to overload than resMgt; this is because PowerTrust fails to consider load in provider selection. However, resMgt has a higher failure rate due to provider unwillingness than PowerTrust because resMgt only considers node load but neglects reputation. Only Harmony can constrain failures due to either cause, as it jointly considers both load and reputation.

4.2 Multi-QoS-oriented Resource Selection

We then evaluate the effectiveness of the multi-QoS-oriented resource selection scheme. We used 95×12 transaction records for 95 sellers, each with 12 types of merchandise. Each transaction record has 52 transactions. We used 40×12 for training and the remaining 55×12 for testing. We regard a node's individual reputation (rating on its transactions for a type of merchandise) as its overall QoS for the merchandise (i.e., resource) and regard its overall reputation as its reputation in Harmony. The inputs of the neural network model include the QoS attributes in each transaction (i.e., price, distance, service, quality and efficiency) and the seller's overall reputation. The output of the model is the seller's overall QoS. Because the real trace does not have users' consideration priorities, we assume that the six QoS attributes have equal priorities. Figure 11 shows the predicted overall QoS and the real overall QoS for 100 resource requests, both of which almost overlap. Their root mean square error equals 0.95, a very small value. The results show the effectiveness and accuracy of the neural network model in predicting the QoS in individual resource selection.

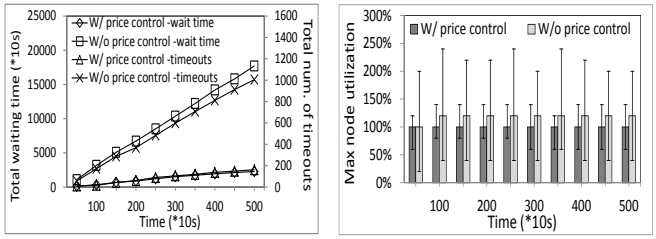
4.3 Effect of Queuing Timeout

In order to study the effect of the timeout for dropping requests on the success rate and failure rate, we measured the two metrics with varying timeout values for 5000 requests. Figure 12(a) shows the success rates of different methods with different timeout values. We see that Harmony and resMgt generate no delayed requests since they choose lightly loaded nodes, while PowerTrust generates many delayed successful requests since it does not consider node load. This result is consistent with that in Figure 10(a). We observe that the timeout value does not affect the success rate of Harmony and resMgt. Also, as the timeout value increases, the success



(a) Ave. success rate (b) Ave. failure rate

Fig. 12. Effect of queuing timeout on resource sharing.



(a) Wait time and timeouts (b) Max. node utilization

Fig. 13. Effectiveness of price-assisted control algorithm.

rate of PowerTrust for delayed successful requests increases. As Harmony and resMgt always choose lightly loaded nodes, they do not need to store requests that cannot be processed in time into a queue. PowerTrust heavily depends on the queue since it is very likely to choose an overloaded node. A smaller timeout introduces more request drops and hence fewer delayed successful requests, and vice versa. We also observe that the success rate follows Harmony>resMgt>PowerTrust except when the timeout equals 1000s for the same reasons as in Figure 10(a). The large timeout of 1000s allows PowerTrust to store and resolve more requests, thus producing more successful requests than resMgt.

Figure 12(b) shows the failure rate due to overload or unwillingness for the three methods at different timeouts. We see that the failures in Harmony and resMgt are all caused by provider unwillingness to provide services, and the failures in PowerTrust are caused by both overload and unwillingness. As the timeout value increases, the failure rate caused by provider overload decreases. A larger timeout enables more requests to be resolved and leads to few failures and vice versa. These results are consistent with those in Figure 12(a). We also observe that the failure rate follows Harmony<resMgt<PowerTrust except when the timeout equals 1000s, for the same reasons as in Figure 10(c). The large 1000s timeout allows PowerTrust to store and resolve more requests, thus producing fewer failed requests than resMgt. In conclusion, Harmony and resMgt are not affected by the timeout, while PowerTrust is sensitive to the timeout and a large timeout enables it to resolve more requests. Moreover, Harmony always achieves a higher success rate than the other methods since it considers both reputation and load status.

4.4 Price-assisted Resource/Reputation Control

We then test the performance on a system with and without the price-assisted resource/reputation control algorithm denoted by *w/Price* and *w/oPrice*, respectively, in a heavy load situation. We randomly chose nodes from the system to generate requests. The request generating rate follows a Poisson process at a rate of 2 requests per second. We set every request to use 40 units of a resource for 200s. We chose these values so that some resource providers have insufficient available resources when requested. We set the price range to [10, 20] credits. All nodes have the same capacity of 200 and reputation of 9. During the simulation period, no resource information is updated in the directory nodes. To choose a resource provider, a requester first identifies the providers with reputation > 8, then identifies the providers with the lowest price, and finally randomly chooses one. Nodes with reputation > 8 have probability 1 to offer the service. A resource request fails only when its provider is overloaded. If a requester receives a successful service, the resource provider's reputation is increased by 0.001. Otherwise, its reputation is decreased by 0.02. In every 10s, each node checks its load; if its load factor $f > 0.8$, it reduces its price by 1; otherwise, it increases its price by 1. Also, each provider charges its clients for its offered resources and its reputation is updated in every 10s. The timeout of the waiting queue of each provider was set to 60s. We randomly determined the values of the above parameters in their reasonable ranges, and different parameter values should not change the relative performance differences between different methods.

Figure 13(a) shows the total request waiting time in node queues and total number of request timeouts versus time. *w/oPrice* generates a significantly higher total request waiting time than *w/Price*. Also, the request waiting time of *w/oPrice* increases dramatically while that of *w/Price* grows only slightly as time goes on. This is because nodes in *w/Price* adjust their own load by price, which can effectively prevent node overload and avoid long queues. In contrast, reputed nodes in *w/oPrice* are likely to be overloaded since they have a high probability of being chosen as providers but have no strategy to adjust their load. Due to the same reason, the total number of request timeouts of *w/oPrice* is much smaller than that of *w/Price*.

Figure 13(b) shows the median, 99th and 1st percentiles of all nodes' maximum utilizations every 500s. We see that *w/Price* generates smaller median values, much smaller 99th percentile values, and larger 1st percentile values than *w/oPrice*. These results imply that *w/Price* distributes request load among servers more evenly than *w/oPrice*. *w/Price* makes full use of all available resources while constraining node utilization within 100%; while in *w/oPrice*, some nodes are overloaded or underloaded. The experimental results in both figures further verify the effectiveness of the price-based control algorithm in fully utilizing available resources and preventing overloads.

5 Conclusion

In this paper, we propose an integrated resource/reputation management platform, called Harmony, for collaborative cloud computing (CCC). Recognizing the interdependencies between resource management and reputation management, Harmony incorporates three innovative components to enhance their mutual interactions for efficient and trustworthy resource sharing among clouds. *The integrated resource/reputation management* component efficiently and effectively collects and provides information about available resources and reputations of providers for providing the types of resources. *The multi-QoS-oriented resource selection* component helps requesters choose resource providers that offer the highest QoS measured by the requesters' priority consideration of multiple QoS attributes. *The price-assisted resource/reputation control* component provides incentives for nodes to offer high QoS in providing resources. Also, it helps providers keep their high reputations and avoid being overloaded while maximizing incomes. The components collaborate to enhance the efficiency and reliability of sharing globally-scattered distributed resources in CCC. Simulations and trace-driven experiments on PlanetLab verify the effectiveness of the different Harmony components and the superior performance of Harmony in comparison to previous resource and reputation management systems. The experimental results also show that Harmony achieves high scalability, balanced load distribution, locality-awareness, and dynamism-resilience in the large-scale and

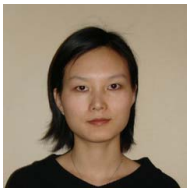
dynamic CCC environment. In our future work, we will investigate the optimal time period for neural network training and load factor calculation. We will investigate the challenges of deploying the Harmony system for the real-world applications which involve cooperation between cloud providers.

Acknowledgment

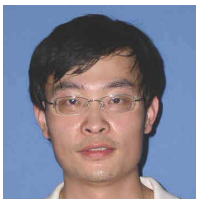
This research was supported in part by U.S. NSF grants OCI-1064230, CNS-1049947, CNS-1156875, CNS-0917056 and CNS-1057530, CNS-1025652, CNS-0938189, Microsoft Research Faculty Fellowship 8300751, and Sandia National Laboratories grant 10002282. Early versions of this work [15, 16] were presented in the Proceedings of ICCCN'11 and NAS'08 (the best paper).

References

[1] Amazon elastic compute cloud (EC2). http://aws.amazon.com.
[2] Dropbox. Available: www.dropbox.com.
[3] P. Suresh Kumar, P. Sateesh Kumar, and S. Ramachandram. Recent Trust Models In Grid. *JATIT*, 2011.
[4] J. Li, B. Li, Z. Du, and L. Meng. CloudVO: Building a Secure Virtual Organization for Multiple Clouds Collaboration. In *Proc. of SNPD*, 2010.
[5] C. Liu, B. T. Loo, and Y. Mao. Declarative Automated Cloud Resource Orchestration. In *Proc. of SOCC*, 2011.
[6] C. Liu, Y. Mao, J. E. Van der Merwe, and M. F. Fernandez. Cloud Resource Orchestration: A DataCentric Approach. In *Proc. of CIDR*, 2011.
[7] K. Hwang, S. Kulkarni, and Y. Hu. Cloud Security with Virtualized Defense and Reputation-based Trust Management. In *Proc. of DASC*, 2009.
[8] IBM Red Boo. Fundamentals of Grid Computing. Technical Report REDP-3613-00, 2000.
[9] L. Xiong and L. Liu. Peertrust: Supporting Reputation-based Trust for Peer-to-Peer Electronic Communities. *TKDE*, 2004.
[10] M. Srivatsa, L. Xiong, and L. Liu. Trustguard: Countering Vulnerabilities in Reputation Management for Decentralized Overlay Networks. In *Proc. of World Wide Web Conference*, 2005.
[11] R. Zhou and K. Hwang. PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing. *IEEE TPDS*, 2008.
[12] R. Zhou and K. Hwang. Gossip-based reputation management for unstructured peer-to-peer networks. *TKDE*, 2007.
[13] Zol. http://www.zol.com.cn/.
[14] PlanetLab. http://www.planet-lab.org/.
[15] H. Shen and G. Liu. Harmony: Integrated resource and reputation management for large-scale distributed systems. In *Proc. of ICCCN*, 2011.
[16] H. Shen, Y. Zhu, and W. Li. Efficient and Locality-aware Resource Management in Wide-area Distributed Systems. In *Proc. of NAS*, 2008.
[17] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *Proc. of SIGCOMM*, 2004.
[18] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. In *Proc. of SIGCOMM*, 2003.
[19] M. Cai, M. Frank, and P. Szekely. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. *Grid Computing*, 2004.
[20] D. Talia, P. Trunfio, J. Zeng, and M. Högqvist. A DHT-based Peer-to-Peer Framework for Resource Discovery in Grids. Technical Report TR-0048, CoreGRID - Network of Excellence, 2006.
[21] H. Shen, C. Xu, and G. Chen. Cycloid: A Scalable Constant-Degree P2P Overlay Network. *Performance Evaluation*, 63(3):195–216, 2006.
[22] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proc. of STOC*, pages 654–663, 1997.
[23] H. Shen and C. Xu. Hash-based proximity clustering for efficient load balancing in heterogeneous DHT networks. *JPDC*, 2008.
[24] H. Shen and C.-Z. Xu. Hash-based Proximity Clustering for Efficient Load Balancing in Heterogeneous DHT Networks. *JPDC*, 2008.
[25] M. Cai and K. Hwang. Distributed Aggregation Algorithms with Load-Balancing for Scalable Grid Resource Monitoring. In *Proc. of IPDPS*, 2007.
[26] H. Shen and C.-Z. Xu. Leveraging a Compound Graph based DHT for Multi-Attribute Range Queries with Performance Analysis. *TC*, 2011.
[27] H. Shen and K. Hwang. Locality-Preserving Clustering and Discovery of Resources in Wide-Area Distributed Computational Grids. *TC*, 2011.
[28] H. Shen. A P2P-based Intelligent Resource Discovery Mechanism in Internet-based Distributed Systems. *JPDC*, 2008.
[29] H. Shen and C. Xu. Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured Peer-to-Peer Networks. *TPDS*, 2007.
[30] S. Haykin, editor. *Neural Networks: A Comprehensive Foundataion*. Second Edition, Prentice-Hall Publisher, 1999.
[31] E. A. Wan. Neural network classification: a Bayesian interpretation. *TNN*, 1990.
[32] M. Mowbray, F. Brasileiro, N. Andrade, and J. Santana. A Reciprocation-Based Economy for Multiple Services in Peer-to-Peer Grids. In *Proc. of P2P*, 2006.
[33] S. C. M. Lee, J. W. J. Jiang, D.-M. Chiu, and J. C. S. Lui. Interaction of ISPs: Distributed Resource Allocation and Revenue Maximization. *TPDS*, 19(2):204–218, 2008.
[34] J. Park and M. van der Schaar. Pricing and Incentives in Peer-to-Peer Networks. In *Proc. of INFOCOM*, 2010.
[35] X. Zhang and B. Li. On the Market Power of Network Coding in P2P Content Distribution Systems. In *Proc. of INFOCOM*, 2009.
[36] P. J. Welch and G. F. Welch. Economics: Theory and Practice. *Wiley, ISBN-10: 0471679461*, 2006.
[37] N. Bansal and M. Harchol-Balter. Analysis of srpt scheduling: investigating unfairness. In *SIGMETRICS/Performance*, 2001.
[38] X. Zhang, Y. Qu, and L. Xiao. Improving distributed workload performance by sharing both cpu and memory resources. In *Proc. of ICDCS*, 2000.
[39] R. Subrata and A. Y. Zomaya. Game-theoretic approach for load balancing in computational grids. *TPDS*, 2008.
[40] D. Qiu and R. Srikant. Modeling and performance analysis of Bittorrent-like Peer-to-Peer networks. In *Proc. of SIGCOMM*, 2004.
[41] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of INFOCOM*, 1996.
[42] K.-C. Lai, K.-C Huang, C.-S. Koong, Y.-F. Yu, P.-J. Huang, Q.-J. Chen, and T.-. Hunang. A P2P Resource Discovery Strategy for Cloud Computing Systems. *Journal of Computers*, 2010.
[43] I. Al-Oqily and A. Karmouch. SORD: A Fault-Resilient Service Overlay for MediaPort Resource Discovery. *TPDS*, 2008.
[44] S. Lee, X. Ren, and R. Eigenmann. Efficient Content Search In Ishare, A P2P Based Internet-Sharing System. In *Proc. of IPDPS*, 2008.
[45] A. El-Atawy Y. Yan and E. Al-Shaer. Ranking-based optimal resource allocation in peer-to-peer networks. In *Proc. of ICDCS*, 2007.
[46] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu. Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures. In *Proc. of ICDCS*, 2010.
[47] M. Cardosa and A. Chandra. Resource Bundles: Using Aggregation for Statistical Large-Scale Resource Discovery and Management. *TPDS*, 2010.
[48] S. Di and C. Wang. Dynamic Optimization of Multiattribute Resource Allocation in Self-Organizing Clouds. *TPDS*, 2013.
[49] I. Konstantinou, D. Tsoumakos, and N. Koziris. Fast and Cost-Effective Online Load-Balancing in Distributed Range-Queriable Systems. *TPDS*, 2011.
[50] Y. Wang and J. Vassileva. Trust and Reputation Model in Peer-to-Peer Networks. In *Proc. of P2P*, 2003.
[51] K. Hwang and D. Li. Trusted Cloud Computing with Secure Resources and Data Coloring. *Internet Computing*, 2010.
[52] P. Srivaramangai and R. Srinivasan. A Comprehensive Trust Model for Improved Reliability in Grid. *IJCA*, 2010.
[53] K. Wei and S. Tang. A Multi-level Trust Evaluation Model Based on D-S Theory for Grid. In *Proc. of CIS*, 2009.
[54] K. Chen, K. W. Hwang, and G. Chen. Heuristic Discovery of Role-Based Trust Chains in Peer-to-Peer Networks. *TPDS*, 2009.
[55] X. Li, F. Zhou, and X. Yang. Scalable Feedback Aggregating (SFA) Overlay for Large-Scale P2P Trust Management. *TPDS*, 2012.
[56] A. Satsiou and L. Tassiulas. Reputation-Based Resource Allocation in P2P Systems of Rational Users. *TPDS*, 2010.
[57] K. Chard and K. Bubendorfer. High Performance Resource Allocation Strategies for Computational Economies. *TPDS*, 2013.
[58] V. Kantere, D. Dash, G. Francois, S. Kyriakopoulou, and A. Ailamaki. Optimal Service Pricing for a Cloud Cache. *TKDE*, 2011.
[59] S. Son and K. M. Sim. A Price- and-Time-Slot-Negotiation Mechanism for Cloud Service Reservations. *TSMC*, 2012.
[60] H. Han, Y. C. Lee, W. Shin, H. Jung, H. Y. Yeom, and A. Y. Zomaya. Cashing in on the Cache in the Cloud. *TPDS*, 2012.
[61] S. Chaisiri, B. S. Lee, and D. Niyato. Optimization of Resource Provisioning Cost in Cloud Computing. *TSC*, 2012.
[62] S. Yi, A. Andrzejak, and D. Kondo. Monetary Cost-Aware Checkpointing and Migration on Amazon Cloud Spot Instances. *TSC*, 2012.
[63] Q. Zhu and G. N. Agrawal. Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments. *TSC*, 2012.



Haiying Shen Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Assistant Professor in the ECE Department at Clemson University. Her research interests include distributed computer systems and computer networks with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and grid and cloud computing. She was the Program Co-Chair for a number of international conferences and member of the Program Committees of many leading conferences. She is a Microsoft Faculty Fellow of 2010 and a member of the IEEE and ACM.



Guoxin Liu Guoxin Liu received the BS degree in BeiHang University 2006, and the MS degree in Institute of Software, Chinese Academy of Sciences 2009. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include distributed networks, with an emphasis on Peer-to-Peer, data center and online social networks. He is a student member of IEEE.

6 Pseudocode for Two Algorithms

Algorithm 1 Pseudocode for resource information storing and retrieving conducted by a node.

```
1:  procedure STORE(resource)
2:      Calculate its Hilbert Number H_i
3:      Calculate the consistent hash value of the resource H_r
4:      Generate resource ID (H_i, H_r)
5:      Use DHT function Insert((H_i, H_r), I_r) to store the resource
6:      information in its local cluster
7:  end procedure
8:
9:  procedure REQUEST(resource)
10:     Calculate its Hilbert Number H_i
11:     Calculate the consistent hash value of the resource H_r
12:     Generate resource ID (H_i, H_r)
13:     Use DHT function Lookup(H_i, H_r) to request the resource
14:     information < ip_addr(j)... >
15:     Request resources from resource provider ip_addr(j)
16: end procedure
17:
18: procedure PROCESS(resourceRequest)
19:     Check its resource directory
20:     if has resource information of requested resource then
21:         Return the resource information
22:     else
23:         while not found the requested resource information do
24:             Probe other nodes in its cluster using the 2-way randomized
25:             probing method
26:         end while
27:     end if
28:     Return the resource information or a search failure response
29: end procedure
```

7 Additional PlanetLab Experimental Results

7.1 Price-assisted Resource/Reputation Control

In this experiment, we set the price range to [10, 20] credits. We tested 39 resource providers in a geographical region for one resource. We randomly chose three providers. One node offers the lowest price (10 credits), one offers the highest price (20 credits), and the other one (called the Harmony node) executes the price-assisted resource/reputation control scheme in Harmony. The remaining nodes offer a medium price (15 credits). All nodes have the same capacity of 200 and reputation of 9. Resource requests were generated once every 10s for the resource in this region. We set every request to use 40 units of a resource for 100s in order to make some resource providers have insufficient available resources when requested. During the simulation period, no resource information is updated in the directory nodes. To choose a resource provider, a requester first identifies the providers with reputation > 8, then identifies the providers with the lowest price, and finally randomly chooses one. Nodes with reputation > 8 have probability 1 to offer the service. A resource request fails only when its provider is overloaded. If a requester receives a successful service, the resource provider's reputation is increased by 0.001. Otherwise, its reputation is decreased by 0.02. In every 10s, the Harmony node checks its load; if its load factor $f > 0.8$, it reduces its price by 1; otherwise, it increases its price by 1. Also, each provider charges its clients for its offered resources and its reputation is updated in every 10s. The timeout of the waiting queue of each provider was set to 60s. We randomly determined the values of the above parameters in their reasonable ranges, and different parameter values should not change the relative performance differences between different methods.

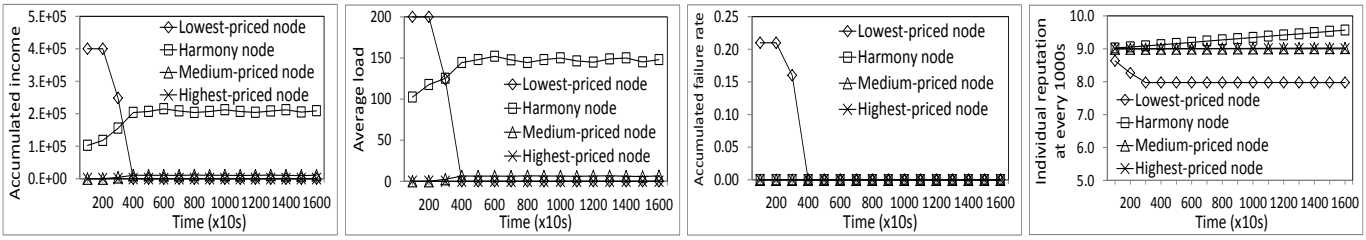
Figures 14(a), 14(b), 14(c), and 14(d) show the accumulated income, average load per node, accumulated failure rate, and reputation for the highest-priced, lowest-priced, medium-priced, and Harmony nodes at every 1,000s. The lowest-priced node begins the experiment with a very high load, income, and failure rate; this clearly is because its price makes it an attractive provider. As it becomes overloaded with requests, however, its failure rate increases, which causes its reputation to fall below 8 and receive no more requests. Its income and load correspondingly plummet. The medium-priced node is unlikely to receive requests, as there are many other nodes with similar prices and reputation, and most of customers are attracted by Harmony node. As such, its income, load, failure rate, and reputation stay relatively constant. Similarly, the highest-priced node is an unattractive provider due to the cost of its resources. It receives no requests, and thus its income, load, and failure rate remain 0. The Harmony node outperforms all other nodes. Its income and load start out small, but increase until they remain stable. Further, it produces no failures and its reputation climbs over the course of the experiment. Since the Harmony node adapts its price to increase or reduce load when needed, other nodes select it as a provider only when it has available resources.

Algorithm 2 The pseudocode of the algorithm executed by node i in Harmony.

```
1:  procedure MULTI-FACETED_RESOURCE/REPUATION_MANAGEMENT
2:      Periodically send Insert((H_i, H_r),I_r) to report its available
3:      resources to the directory node
4:      if need a resource(s) then
5:          for each resource type of needed resource(s) do
6:              Use Lookup(H_i,H_r) function to send the request to the
7:              resource's directory node
8:          end for
9:      end if
10:     if receive a response from a directory node then
11:         for each provider in the response do
12:             Send a request to the provider
13:         end for
14:     end if
15: end procedure
16:
17: procedure MULTI-QOS-ORIENTED_RESOURCE_SELECTION
18:     if it is a directory node then
19:         Build a neural network model for QoS calculation
20:         Periodically train the neural network model
21:         if receive a resource request then
22:             Identify the resource providers satisfying the required amount,
23:             reputation, resource availability time and price
24:             Calculate QoS using the neural network
25:             Choose the providers with the highest QoS
26:             Notify the requester with the selected providers
27:         end if
28:     end if
29: end procedure
30:
31: procedure PRICE-ASSISTED_RESOURCE/REPUTATION_CONTROL
32:     Periodically calculate its load factor f = l/c
33:     if f > α (0.8 ≤ α < 1) then
34:         Increase its price by one price unit
35:     else
36:         Decrease its price by one price unit
37:     end if
38:     if provide other nodes with resources then
39:         Charge credits from each resource receiver
40:     end if
41:     if receive resources from other nodes then
42:         Pay credits to each resource provider
43:     end if
44: end procedure
```

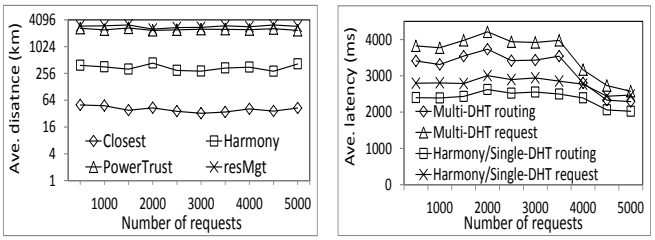
7.2 Efficiency of Resource Location

We randomly chose 12 nodes distributed all over the world. Each of the 12 nodes sends a request every 15s. Because of resource limitations on PlanetLab, we assumed Multi-DHT uses 3 DHTs to manage 12 types of resources, with each DHT managing 4 resources. Figure 15(a) shows the average client-server distances of each system. We also include the optimal result for the case in which each client is able to choose the geographically closest server, denoted by Closest. The figure



(a) Accumulated income (b) Ave. load (c) Accumulated failure rate (d) Ave. reputation

Fig. 14. Effectiveness of Harmony's price-assisted resource/reputation control.



(a) Ave. client-server distance (b) Ave. latency

Fig. 15. Efficiency of resource location.

shows that the average distance of Harmony remains closer to that of Closest than those of resMgt and PowerTrust. The result demonstrates the effectiveness of Harmony in locality-aware resource discovery. ResMgt and PowerTrust generate much longer client-server distances, approximately 10 times that of Harmony, due to their neglect of locality in resource location. In contrast, Harmony enables a node to find geographically close resource providers. Short communication distances help reduce communication latency and cost, thus enhancing the entire system performance.

Figure 15(b) shows the latency in discovering resource providers and requesting resources from the selected providers, respectively. The former is the time elapsed from when a node sends out a request to when it receives a response from a directory node, and the latter is the time elapsed from sending a message to the selected resource provider to receiving the response. To show comparable performance with the same lookup path length, we use Harmony to represent Single-DHT and let Multi-DHT use the same topology and server selection method as Harmony. Since the multiple requests are forwarded simultaneously, it is intriguing to see that Harmony leads to lower latency than Multi-DHT in both resource discovery and resource requesting. This is caused by the heavier traffic generated by more messages in Multi-DHT. A node needs to send out multiple messages for a request with multiple types of resources and the nodes in Multi-DHT need to maintain multiple DHTs with periodical message exchanges. Due to the heavy traffic, some messages cannot be forwarded in time, leading to a longer delay.

7.3 Dynamism-resilience and Maintenance Overhead

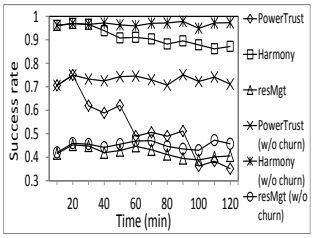
This experiment tests the dynamism-resilience of the systems in node departures. Recall that before a node leaves the system, it notifies the directory nodes that store its resource information. In order to see the impact of the notification policy, we tested the average success rates with and without this policy when two nodes depart from the system every 10 minutes, as shown in Figure 16(a) and Figure 16(b). We also show the results with no node departures in the figures for reference (marked by w/o churn). Each of 22 randomly chose nodes continually generated a request every 15s. Both figures show that the success rate follows Harmony>PowerTrust>resMgt due to the same reason as in Figure 9(a). Figure 16(a) further shows that the success rates of Harmony and resMgt decrease slightly due to node departures, while that of PowerTrust decreases greatly. This is because a request fails to be served if a departed node is chosen as a server. PowerTrust is biased toward the highest-overall-reputed node; if this node for a resource departs, many requests for the resource fail. In contrast, Harmony has many options for the highest-individual reputation nodes, so the probability of choosing a departing node is lower. Similarly, the probability of resMgt choosing a departing node is low. The result verifies the dynamism-resilience of Harmony.

Figure 16(b) illustrates that the success rate of each system with churn remains approximately the same as that without churn. This is because with the notification policy, directory nodes of the resources of a departing node can update their resource information once receiving the notification. The figure also shows that the success rate of Harmony remains nearly constant, while those of PowerTrust and resMgt fluctuate even without churn. This is due to the same reason as in Figure 16(a).

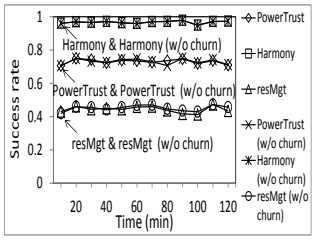
We then test the overhead of the strategy for handling churn when here was one node departure per minute. Figure 17 shows the total number of notification messages for every 10 departures as time goes on. We see that the maintenance overhead for node departures follows Harmony<Single-DHT<Multi-DHT. Since Multi-DHT maintains multiple DHTs; each for one resource type, its number of exchanged messages is multiple times that of Single-DHT. The number of messages per node for one departure notification is $\log n$ in Single-DHT, while it is 7 in Harmony. Thus, Harmony generates lower overhead for node departure notification.

7.4 Load Balancing on Directory Nodes

To avoid being overloaded due to the load of resource reports and queries, a directory node can have several backup nodes with a copy of the reported resource information. When a directory node is overloaded, it redirects a request to its backup node. If this backup node is also overloaded, it re-directs the request to another backup node, until the request arrives at a lightly loaded backup node. We use "Harmony+LoadBalancer" to denote Harmony with this load balancing algorithm. In this experiment, each directory node has 2 backup nodes that are physically close to the directory node. Each request queries for one resource. The capacity of each node follows a Pareto distribution with shape 2 and scale parameter equals 20. Each request needs 2s to process in a lightly loaded node, and 10s in an overloaded node. As in [40], the request initiation follows a Poisson process with the arrival rate equals 20 requests per second. The experiment ran for 2500s. Figure 18 shows the number of requests handled by overloaded directory nodes with or without the load balancing algorithm in Harmony with different number of resource requests, and the number of redirection operations with different numbers of resource requests. We see that without the load balancing algorithm, the number of delayed requests increases when more requests are generated. With this algorithm, Harmony+LoadBalancer produces 0 delayed requests. The figure also shows that the number of redirection operations remains



(a) Ave. success rate w/o notification



(b) Ave. success rate w/ notification

Fig. 16. Dynamism-resilience performance in churn.

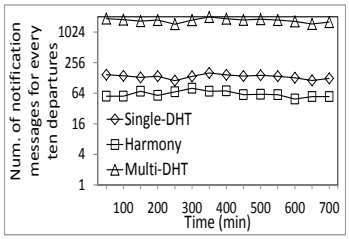


Fig. 17. Cost for handling churn.

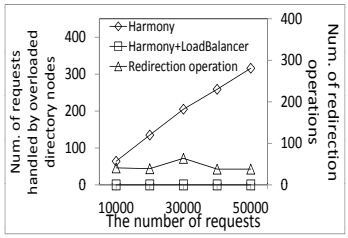
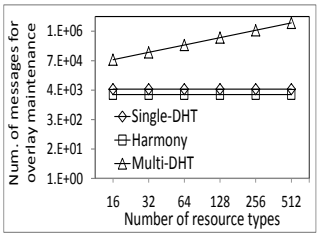
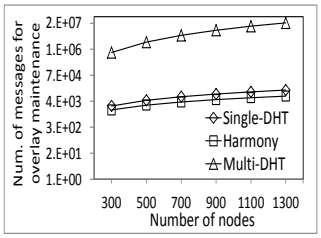


Fig. 18. Load balancing on directory nodes.



(a) Maintenance overhead vs. # of resource types



(b) Maintenance overhead vs. # of nodes

Fig. 19. Maintenance overhead.

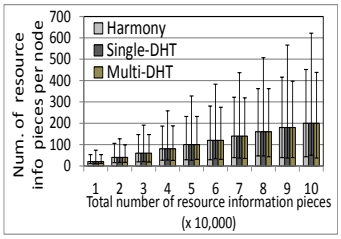


Fig. 20. Load distribution.

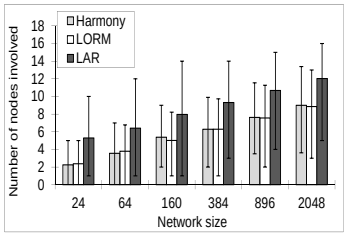


Fig. 21. Number of nodes involved.

constant at around 47. From the experimental results, we found that 97% of the redirections are one hop redirections, which means that one backup node is sufficient to handle the load in the experiment. The results confirm the effectiveness of the load balancing algorithm in Harmony to prevent overloading the directory nodes and avoid delayed requests.

8 Simulation Performance Evaluation

8.1 Maintenance Overhead

Due to the limited number of hosts in PlanetLab, we developed a simulator in order to test the performance of Harmony in a large-scale system with many more nodes. In this experiment, we first set the number of nodes to 500 and the number of resource types in the system to $2^x (x \in [4, 9])$. Figure 19(a) plots the maintenance overhead versus the number of resource types. We see that the maintenance overhead follows Harmony<Single-DHT<Multi-DHT. Recall that each node has 7 neighbors in Cycloid while it has $\log n \approx 9$ neighbors in Chord. Therefore, a node in Harmony needs to send fewer messages to its neighbors than in Single-DHT. Multi-DHT has one DHT for one resource type. In Multi-DHT with m DHTs, a node needs to send $m \log n$ messages, leading to significantly higher overhead. This is also the reason that the maintenance overheads of Harmony and Single-DHT remain constant, but that of Multi-DHT increases as the number of resource types increases.

Figure 19(b) demonstrates the maintenance overhead when the number of nodes in the system was varied from 300 to 1300 with an increase of 200 in each step, and the number of resource types was set to 64. We observe that the maintenance overhead of the three systems increases as the number of nodes increases. In a larger-scale system, each node has more neighbors and needs to send more messages in one stabilization operation. Also, more nodes in the system leads to more message exchanges in stabilization. We notice that the maintenance overhead of Multi-DHT grows faster than others because of its multiple DHTs. Also, the result shows that Harmony<Single-DHT<Multi-DHT due to the same reason as in Figure 19(a).

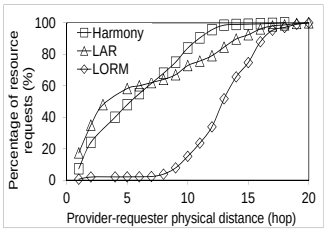
8.2 Load Balance

Each directory node needs to store resource information, search for resource providers in its directory, and reply to requesters. We test whether a system can reach load balance in resource management load distribution among nodes. We refer to a piece of resource information about a provider's available resources in a directory node as one resource piece. We measured the number of resource pieces in each directory node after a number of resource pieces were distributed among nodes. We set the number of nodes to 512, and varied the total number of resource pieces from 10^4 to 10^5 with an increment of 10^4 in each step. The number of resource types in each information piece is randomly chosen from [1,512]. Figure 20 plots the median, the 1st and 99th percentiles of the number of resource information pieces per node. We see that the number of resource information pieces per node increases linearly with the number of resource information pieces in the system. The result of Single-DHT exhibits greater variation than Harmony and Multi-DHT. This is because Single-DHT assigns all resource information pieces of the same type into one node. The information of one resource type is distributed among all nodes in the system in Multi-DHT, and among all nodes in a cluster in Harmony, achieving a more balanced load distribution.

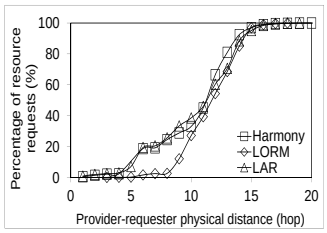
8.3 Efficiency

We also compared Harmony with LAR [29] and LORM [26]. We modified the LAR locality-aware load balancing method for one resource to locality-aware multi-resource management. LAR stores the resource information of physically close nodes into a cluster in Cycloid, and further distributes the information among cluster nodes based on resource type. LAR searches for resources by intra-cluster searching and then inter-cluster randomized probing. Similar to Harmony, LORM enables multiple resource management on one Cycloid DHT. It maps resource type and value to the two levels in the hierarchical Cycloid. Though it provides range querying, it cannot consider locality in resource location.

Figure 21 shows the 1st percentile, 99th percentile and the average number of nodes involved in a resource location operation, which includes message routing nodes and probed nodes, as a function of network size with $n = d \cdot 2^d$ nodes with the dimension d varied from 3 to 8. The number of requests was set to the twice the number of nodes. We can see that the numbers of nodes involved in Harmony and LORM are approximately the same, and less than that in LAR. Since all of them are built on Cycloid, the number of nodes for forwarding a message from a requester to the directory node is the same in all methods. LAR incurs much more involved

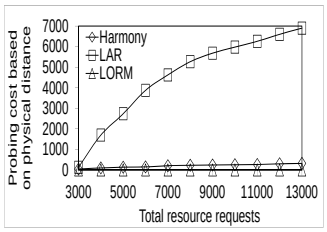


(a) ts5k-large

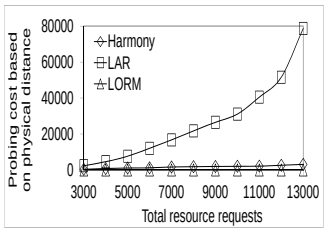


(b) ts5k-small

Fig. 22. CDF of total resource requests distribution.



(a) ts5k-large



(b) ts5k-small

Fig. 23. Physical probing cost.

nodes because it probes nodes for required resources in a system-wide manner. It groups the information of different physically close resources in one cluster. Therefore, when a cluster does not have the information of a requested resource, the nodes in the other clusters should be probed, leading to a large number of involved nodes. In contrast, Harmony and LORM group each type of resource in one cluster. Thus, if a cluster does not have the information of a requested resource, there is no need to probe nodes in other clusters. The probing scope reduction from system-wide to cluster-wide helps to reduce the number of nodes involved, resulting in less overhead and high efficiency.

8.4 Locality-awareness

The number of nodes was set to 4,096 with $d = 11$, and number of resource types was set to 100.The resource requesters were randomly chosen. We use transit-stub topologies generated by GT-ITM [41]: "ts5k-large" and "ts5k-small" with approximately 5,000 nodes each. "ts5k-large" has 5 transit domains, 3 transit nodes per transit domain, 5 stub domains attached to each transit node, and 60 nodes in each stub domain on average. "ts5k-small" has 120 transit domains, 5 transit nodes per transit domain, 4 stub domains attached to each transit node, and 2 nodes in each stub domain on average. "ts5k-large" has a larger backbone and sparser edge network (stub), and it is used to represent a situation in which a CCC system consists of nodes from several big stub domains. "ts5k-small" represents a situation in which a CCC system consists of nodes scattered in the entire Internet and only a few nodes from the same edge network join the overlay. To account for the fact that interdomain routes have higher latency, each interdomain hop counts as 3 hops of latency units while each intradomain hop counts as 1 hop of latency unit.

In the experiment, we randomly generated 10,000 resource requests and recorded the distance between the resource providers and the resource requesters of each request. Figure 22(a) and (b) show the cumulative distribution function (CDF) of the percentage of resource requests versus the distance between resource requesters and providers of different resource management schemes in "ts5k-small" and "ts5k-large," respectively. We can see that in "ts5k-large," 90% and 80% of requests in Harmony and LAR, respectively, find resources within 11 hops, while only about 15% of requests find resources within 10 hops in LORM. Almost all requests find resources within 15 hops in Harmony and LAR, while only 75% of the requests find resources within 15 hops in LORM. The results show that Harmony and LAR can allocate most resources within short distances from requesters while LORM allocates most resources in long distances. It demonstrates the high locality-aware performance of Harmony and LAR to locate nearby resources for the resource requesters in resource management. The figure also shows that LAR allocates more resources within 7 hops than Harmony, while Harmony allocates more resources between 7 and 15 hops.

We found that in LAR, physically close resource information is partitioned into 416 parts (i.e., clusters), while in Harmony the information is partitioned into at most 11 parts based on proximity closeness, because each cluster has at most 11 nodes. Fine-grained resource information in LAR leads to higher locality-aware performance. However, less information in one directory leads to a lower probability of locating required resources in the directory. Then, the directory node uses randomized probing in the entire system for requested resources in LAR, which may find physically distant resources. This explains why LAR does not perform as well as Harmony in distances within 7 to 15 hops. Figure 22(b) shows the same trends as in "ts5k-large," although the performance difference between mechanisms is not as significant..

8.5 Probing Cost

When a directory node does not have information on required resources, it probes other directory nodes for requested resources. Therefore, resource probing constitutes a main part of resource management overhead. The cost is directly related to message size and physical path length of the message travelled; we use the product of these two factors to represent the cost. It is assumed that the size of a message is one unit. In the experiment, we varied the number of resource requests from 3,000 to 13,000, with a step size of 1,000.

Figures 23(a) and (b) plot the probing cost of Harmony, LAR and LORM in "ts5k-large" and "ts5k-small," respectively. From these figures, we can see that the probing cost of LORM remains at 0, and the costs of Harmony and LAR increase with the number of requests. The cost of LAR grows dramatically faster than Harmony, while Harmony only has a marginal increase. LORM stores the information of a resource within a certain range into one directory node. If the directory node does not have the information of requested resources, then there is no requested resource in the system. Therefore, the directory node does not need to probe other nodes for the resource, leading to zero probing cost. In LAR, the directory node probes nodes in the entire system using randomized-probing. Therefore, the directory node may contact nodes very far away from itself, and a request for a non-existing resource may lead to infinite probing. In addition, the fine-grained resource information in LAR contributes to more directory node probings before the requested resource is located. On the other hand, in Harmony, the directory node only needs to probe other nodes in its cluster rather than nodes in the entire system. Consequently, its reduced probing scope helps to significantly reduce the probing cost. Combined with the experimental results of locality-aware performance, these results demonstrate that Harmony achieves locality-aware resource management at a cost of marginally higher communication overhead than LORM. However, LORM has poor performance in mapping physically close resource requester and providers. LAR is comparable with Harmony in locality-aware performance, but it generates high node communication overhead in probing.

9 Related Work

Resource Management One group of distributed resource management systems are based on DHTs. Some efforts depend on multiple DHTs with each DHT responsible for one resource [17, 20], while other efforts depend on a single DHT

overlay [19, 25]. The former generates high maintenance overhead due to multiple DHTs, while some of the latter methods may generate bottlenecks because one node is responsible for all resource information of one resource. LORM [26] and MARQ [42] use a hierarchical P2P overlay to realize multi-resource range querying. However, few of these approaches can handle locality and dynamism simultaneously. By locality, we mean physically close resource requesters and providers are mapped to achieve high efficiency. HCO [27] and PIRD [28] can overcome these shortcomings. HCO clusters resources so that a node can always locate physically close multiple resources in its neighborhood. PIRD weaves multiple resources to one index for multi-resource discovery. Harmony is based on these previous works. It focuses on the integration of resMgt and repMgt. That is, it enables a node to discover its requested resource and this resource's reputation simultaneously; to select a resource by considering both efficiency and reputation; and to maintain its highly reputed status and fully utilize its resources while avoid being overloaded.

Liu *et al.* [5] presented COPE (Cloud Orchestration Policy Engine), a distributed platform that allows cloud providers to perform declarative automated cloud resource orchestration. Al-Oqily *et al.* [43] proposed a fault-resilient service overlay for MediaPort resource discovery. iShare [44] facilitates the sharing of diverse resources located in different administrative domains over the Internet. It organizes resources into a hierarchical name space, which is distributed over the underlying structured P2P network. Yan *et al.* [45] incorporated peers' behavioral rankings into the resource allocation to provide differentiated services. Jung *et al.* [46] proposed Mistral, a holistic controller framework that optimizes power consumption, performance benefits, and the transient costs incurred by various adaptations and the controller itself to maximize overall utility. Cardosa *et al.* [47] proposed the notion of a resource bundle, a representative resource usage distribution for a group of nodes with similar resource usage patterns, in order to provide statistical guarantees for resource capacities and achieve scalability. Di *et al.* [48] used a proportional share model to maximize resource utilization, and designed a multi-attribute range query protocol for locating qualified nodes efficiently in a gigantic self-organizing cloud. Konstantinou *et al.* [49] proposed a hybrid method that adaptively utilizes iterative key redistribution and node migration to achieve both fast and cost-effective load-balancing in distributed systems that support range queries.

However, few works have exploited the effective usage of a reputation metric for high-QoS resource selection. The joint consideration of a server's reputation and a client's multiple QoS demands is an innovative aspect of Harmony. Different from the previous resource management methods, Harmony enables a node to locate its desired resources and find the reputation of the located resources. This allows nodes to jointly consider resource reputation in resource selection in order to receive high QoS. That is, Harmony enables clients to choose resource providers not only by resource availability but also by the provider's reputation on providing the resource.

Reputation Management Many reputation systems have been proposed aiming to improve scalability or accuracy of reputation calculation. These works include PeerTrust [9], TrustGuard [10], PowerTrust [11] and GossipTrust [12]. Wang *et al.* [50] proposed a flexible method to present differentiated trust, such as file quality and file type in file sharing, and combine different aspects of trust. The concept of differentiated trust shares similarity with the multi-faceted reputation in Harmony. However, Wang's work focuses on presenting different trust aspects of one service and combining them for a global trust, while Harmony focuses on the reputation differentiation and the integration between resMgt and repMgt. Hwang and Li [51] proposed to build a trust-overlay network over multiple data centers to implement a reputation system for establishing trust between service providers and data owners in clouds. Many other trust models [52, 53] have been proposed recently in grid and cloud computing systems. Chen *et al.* [54] proposed a heuristic-weighting approach to selecting the most likely path to construct a role-based trust chain to establish trust among any pair of anonymous peers in P2P networks. Li *et al.* [55] proposed a scalable feedback aggregating overlay to provide an efficient and effective way to build a reputation-based trust relationship among peers in P2P networks. Satsiou *et al.* [56] proposed a distributed reputation-based system according to which peers earn reputation analogous to their contributions. However, all of these reputation systems give a node one global reputation value, and thus are not effective enough to provide precise guidance for trustworthy individual resource selection.

Harmony is the first to integrate resMgt with repMgt for multi-faceted reputation evaluation and trustworthy individual resource selection. Different from previous reputation systems that only provide the overall reputation for a resource provider, Harmony enables nodes to find the reputation of a specific resource by distributing the reputation information of different resources to different node clusters. Therefore, when a node locates its desired resource, it can also know the reputation of the resource, which helps it to select trustworthy resources. Harmony also enables a client to consider multiple QoS criteria such as price, distance, and reputation when selecting resources.

Economic Marketing Approaches Economic marketing approaches have been proposed [32–35] in recent years to provide incentives for cooperation. Mowbray *et al.* [32] studied reciprocation based mechanisms to encourage donation in P2P grids, where multiple services are shared. Lee *et al.* [33] studied the interaction between ISPs at different tiers and provided insightful evidence to show that ISPs can still gain profits as they upgrade their network infrastructures. Jaeok *et al.* [34] investigated incentives in content production and sharing over P2P networks using a game theoretic model. Zhang *et al.* [35] developed an analytical framework that characterizes a coding based P2P content distribution market, where peers selfishly seek for individual payoff maximization. Chard *et al.* [57] addressed the performance limitations of existing economic allocation models by defining strategies to reduce the failure and reallocation rate, increase occupancy and thereby increase the obtainable utilization of the system. Kantere *et al.* [58] proposed a price-demand model designed for a cloud cache and a dynamic pricing scheme for queries executed in the cloud cache. The pricing solution estimates the correlations of the cache services in a time-efficient manner, and maximizes cloud profit while giving guarantees for user satisfaction. Son *et al.* [59] proposed a multi-issue negotiation mechanism that considers the tradeoff between price and time-slot utilities for cloud service reservations. Han *et al.* [60] proposed the cache as a service (CaaS) model as an optional infrastructure service offering disk cache using a large pool of memory. They proposed a efficient novel pricing scheme to benefit both users and providers. Due to the uncertainty of customer resource demand, Chaisiri *et al.* [61] studied how to reduce customer cost by choosing reservation or on-demand plans of resources in cloud and proposed an optimal cloud resource provisioning algorithm to minimize the total customer cost. Yi *et al.* [62] proposed a scheme based on checkpointing and migration to minimize the cost and volatility of resource provisioning in cloud to reduce both monetary costs and task completion time. Zhu *et al.* [63] proposed a framework based on a multi-input-multi-output feedback control model in a cloud computing environment to minimize resource costs

while meeting an application's need.

Harmony is the first effort on leveraging price to jointly control each node's resource load and reputation, leading to system-wide high performance. It can be used to complement the previous economic marketing approaches.