Introducing a Distributed Cloud Architecture with Efficient Resource Discovery and Optimal Resource Allocation

Praveen Khethavath, Johnson Thomas, Eric Chan-Tin and Hong Liu
Computer Science
Oklahoma State University
Email: {khethav, jpt, chantin, liuh}@cs.okstate.edu

***Abstract*—Cloud computing is an emerging field in computer science. Users are utilizing less of their own existing resources, while increasing usage of cloud resources. With the emergence of new technologies such as mobile devices, these devices are usually under-utilized, and can provide similar functionality to a cloud provided they are properly configured and managed. This paper proposes a *Distributed Cloud Architecture* to make use of independent resources provided by the devices/users. Resource discovery and allocation is critical in designing an efficient and practical distributed cloud. We propose using multi-valued distributed hash tables for efficient resource discovery. Leveraging the fact that there are many users providing resources such as CPU and memory, we define these resources under one key to easily locate devices with equivalent resources. We then propose a new auction mechanism, using a reserve bid formulated rationally by each user for the optimal allocation of discovered resources. Then we discuss how the Nash Equilibrium is achieved based on user requirements.**

***Keywords*-Cloud Computing, Distributed Cloud, Auction Game, Resource Allocation, Resource Discovery, Game Theory.**

I. Introduction

Cloud computing [1] is a common terminology used in both business and academic fields. Cloud computing refers to a different way of computing over the Internet where dynamically scaled shared resources are provided as a service to avoid costs of resource over-provisioning [1,2,3]. Many companies are now relying and performing their operations in the "cloud". Current cloud architectures provide both storage and computation services for users. Cloud computing is a combination of several concepts including virtualization, resource pooling, resource monitoring, dynamic provisioning, utility computing, multi-tenancy, and elasticity. The main entities of cloud computing are service providers, physical resources, virtualized resources, and end-users. Existing cloud architectures are centralized; the virtualized resources are hosted on one (or more) physical machines usually located in the same data center. This provides a single point of entry and has the potential to introduce limited bandwidth, higher latency, and increased network traffic. Moreover, as users' computational needs are unknown, the physical resources are "shared" and virtualized among all the users. Thus, current clouds over-provision available resources to meet potential needs. This centralized nature introduces a waste of resources, high energy consumption, and increased distance to end-users.

With more people using cloud services, these people's individual machines which can be a PC, Laptop or a server which is capable of running VM's are under-utilized in terms of resources. These unused resources (for example, idle CPU time and unused memory) can be utilized to perform other people's computations. We introduce the notion of a *distributed cloud*. As far as we know, we are the first to introduce such a system, with an efficient resource discovery and optimal resource allocation algorithm. The distributed cloud overcomes the downsides of existing cloud architectures, as there is no central entry point and all the resources are distributed. There is no waste of over-provisioned resources, and with the distributed nature, it is possible to find a "cloud machine" geographically close to every user. Our proposed distributed cloud is different from geographically-distributed data centers[18] cloud system, as each machine is an individual user's device and not a dedicated central server. Distributed cloud [5] computing refers to the managing and provisioning of distributed resources. We describe the distributed resources provided by users in a completely decentralized fashion, instead of huge data centres located in one or multiple locations. Moreover the proposed system provides a completely decentralized mechanism for discovering and allocating resources.

The distributed cloud should have all the characteristics of existing cloud architectures, including proper management of resources, data security, data privacy, and trust. In addition, the distributed cloud provides scalability and reduces the network constraints, as all the resources are completely decentralized. Moreover, network latency is improved because it is possible to discover a compute node close to every user. The distributed cloud would also be free of cost as end-users share their available resources. There are many challenges in the design of a distributed cloud, such as ensuring integrity of the computation, privacy of the computation, and fair resource sharing. The challenges also include those faced by decentralized systems such as peer-to-peer systems. This includes free riding, bad mouthing, misrouting, and advertising fake resources. The benefits of a distributed cloud are numerous and the deployment of such a system will have huge societal impacts. The distributed cloud can consist of billions of compute nodes, with a very low-cost to a user donating its machine and resources.

In this paper, we introduce the core design of a distributed cloud. We will address two main issues, *efficient resource discovery*, and *optimal resource allocation*, and leave the rest of the challenges for future work.

In a distributed cloud, there are many machines/nodes and each node has different available resources. Each node only knows a subset of all the nodes in the cloud. Knowing all the nodes is infeasible as it would require gathering information about the whole Internet. A user wants to perform a certain computation which requires a certain amount of resources (either processor power or amounts of memory). The user needs to locate enough nodes that match its resource requirements, such that the computation can be completed. We propose a *multi-valued distributed hash table* (DHT) scheme that would allow for an efficient discovery of resources. Existing DHTs [8,9,10] only work for single dimensional queries (one key-value pair). For the proposed distributed cloud, there are multiple queries which can satisfy the user's requirements. The problem of discovering resources in decentralized systems with multiple attributes can be handled using range queries. Many range query schemes[23,24] have been proposed for peer-to-peer and grid systems for data retrieval. Since these range query schemes are built over existing architectures and works by flooding the queries over network using different techniques[24], they introduce a large overhead and can only be used for data management. Thus, existing DHTs cannot be used in the distributed cloud to efficiently discover available nodes with the required resources to perform the desired computation. Computing resources have many fine grained constraints that needs to be considered, such as processor and memory. The machines with resources that fulfill at least the minimum requirements of the user need to be discovered. A subset of these machines can then be used to perform the computation. The proposed multi-valued DHT is more efficient than current known systems for a completely distributed cloud. When a new node joins its information is updated to all the nodes using the kademlia protocol [8].

Other than the discovery of resource, the resource allocation needs to be considered. There are many users wanting to perform various computations. A node resource could be discovered and asked to perform computation by many users. The node has to allocate its resource optimally to serve the maximum number of users. It also has to consider the participation factor of the user and the throughput required for the computation. The resources can be optimally allocated based on the user requirements. Therefore there is a need for an efficient strategic mechanism to allocate resources. Game theory[6,7,21], a strategic decision-making mechanism, can be used for allocating resources optimally such that all users' requirements are met. We introduce an auction model, where providers bid for resource provisioning. By introducing incentives, our model encourages providers to compete for resources and consider participation of users when determining the allocation of resources. The more resources provided by users, the more they can use the other distributed resources in the system. The system will then schedule resources such that the whole distributed cloud's resources are optimally used. The incentive-based auction model is utility driven, where the utility is calculated from user's requirements, such as the participation factor, processor information, and available memory. In the proposed model, there are N users in the system, and each resource can be allocated to only one user at a time. The model will be shown to be optimal and a Nash equilibrium can be achieved. The model can be further extended to include other user requirements.

The rest of the paper is organized as follows. Section II describes the related work about cloud computing and peer-to-peer system, which both constitute major components to a distributed cloud. Section III describes the proposed design for an efficient resource discovery in a completely decentralized system. The game theoretical approach for optimal resource allocation in a distributed cloud environment is described in Section IV. We conclude in Section V and provides further avenues for future work.

II. Related Work

Cloud Computing. Virtualization [4] is a key concept used in cloud computing. Xen [4] is a popular virtualization management tool used by many cloud service providers. Hyper-V, KVM, and Sun xVM are some of the other virtualization management tools commonly used. Different cloud providers use different programming frameworks for their cloud. For example, Amazon uses Amazon Machine Interface, Google uses Map reduce, Sun uses Solaris OS and Java, Azure uses Microsoft.NET, and open clouds such as Eucalyptus and Open Nebula, use Hibernate, Axis2, Java, and Ruby. There is no specific framework or requirements for cloud computing architectures. All cloud providers provide services defined by service models such as Infrastructure as a Service (IaaS), Platform as a Service (Paas), and Software as a Service (SaaS). Currently all cloud providers rely on huge data centers and are predominantly centralized [1, 2, 5, 6].

Distributed Computing. Distributed or grid computing [15,16,17] uses multiple autonomous computers over a network to solve computational problems as one single unit. Allocation of resources in distributed computing to solve a specific task is NP-hard [13, 14]. There have been many models developed to optimize the resource allocation problems in distributed computing. In distributed computing environments, there are many machines distributed around the world. However, there is a central server that contains information about all these machines and schedules computations. The scalability of distributed computing is in the thousands of machines and resources. Although cloud computing and distributed computing share a lot of similarities, a distributed cloud computing architecture requires a different environment to operate efficiently and securely. A distributed cloud can scale to billions of devices and it is infeasible for one machine to learn about all available resources in the cloud. Moreover, multiple users are concurrently scheduling heterogeneous computations.

When it comes to modeling, there is no specific cloud resource description framework which can be used to describe the resources in the cloud (distributed or centralized). Computer networks and their resources can be described using many existing frameworks such as RDF[25] (resource description framework) or the network description language. In a distributed cloud computing environment we need a specific resource description which can be used for identifying the resources and allow users to request resources clearly. This resource description is needed by the distributed cloud to perform resource discovery and resource allocation.

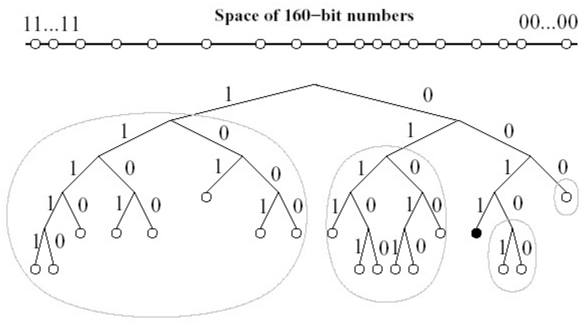


Figure 1. Kademlia routing table.

Peer-to-peer Systems. Content distribution and file sharing was made possible by peer-to-peer systems[11]. Current peer-to-peer systems [8,9,10] use a distributed hash table (DHT) for efficient query lookups. These systems are able to handle churn, node failures, flash crowds, and balance load efficiently. Kademlia [8] is a popular peer-to-peer protocol used by millions of people daily [19,20]. Kademlia has been shown to provide efficient query lookup and publishing; with a total of N peers in the network, a query takes on average $O(logN)$ hops[8] to find the requested peer. Every peer in Kademlia has a 160-bit node ID. The ID is either randomly created or generated from the IP address. Every peer also has a routing table which contains a subset of all the peers in the network. The routing table is a binary tree, as shown in Figure 1. Let's say peer A's node ID is 100101 (actual node ID would be 160 bits long). The black dot in the figure shows a peer B in A's routing table. The first four bits of B's node ID is 1010. The first two bits are the same as A and the next two bits are different. Routing is performed using the bit-wise XOR metric where the XOR between two node IDs gives the distance between thse two peers. Since a peer A only knows a subset of other peers, before it can find another peer (with node ID T), A has to perform a lookup query. Routing steps are illustrated in Figure 2. To improve effectiveness, the routing table is an unbalanced binary tree, where every peer knows more peers closer to itself in terms of XOR distance, than peers further away. Moreover, instead of storing one contact in its routing table at each "leaf", k contacts are stored in a k-bucket.

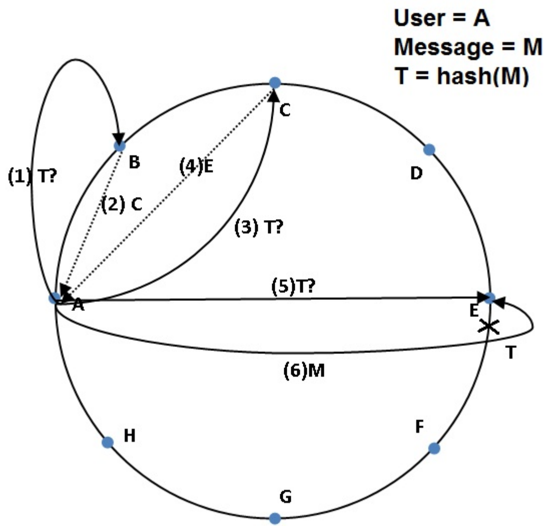


Figure 2. Routing in a distributed hash table steps: 1) User A contacts B, the closest node to T in A's routing table; 2) B replies with C, the closest node to T that B knows about; 3) A contacts C; 4) C replies with E, 5) A contacts E and finds that E is the closest node to T; 6) A sends the request message M to E.

Game Theory. Game theory [6,21], a strategic decision-making methodology, has been extensively used in wireless networks, peer-to-peer systems, grid computing, and in many scientific systems. Game theory is used in many areas such as resource allocation, optimizing and improving decision making. There are many different types of games such as non-cooperative games, cooperative games, Bayesian games, differential games, evolutionary games and auction theory and mechanism design. We can use any of these games based on the requirement and type of information available. There is no work done as far as we know on distributed cloud using game theory. Game theory has been used in existing cloud computing technology for resource allocation [7,26] such that total payment is reduced for the user. In [26] the author talks about a game theoretical resource allocation in existing cloud computing based on tasks its receive and availability of resources such that users utilization is maximized. In existing cloud users need to select resources even before they start using it and instances are already defined in existing cloud. Different types of games are used and cost was the basis for these games. We introduce an incentive-based mechanism and efficient resource allocation in our proposed model of distributed cloud computing.

III. Resource Discovery

The proposed distributed cloud will have geographically distributed resources which will be accessed in a decentralized peer-to-peer fashion. As far as we know, we are the first in designing an efficient resources discovery scheme for computing in a distributed cloud. The distributed resources have many characteristics which should be defined properly. These characteristics would be not only useful for identifying and analysing user requirements properly but would also help to locate resources accurately as needed. The attributes we consider for now are 1) the node ID, which is a unique identifier for each peer, 2) the resource processing power, measured in gigahertz, and 3) the resource of memory available, measured in gigabytes.

A. Naïve Solution

A simple and naïve solution to the problem of resource discovery in a distributed cloud, is making modifications to the original Kademlia algorithm to handle multiple attributes (resources). The node ID can be changed to include the resources information. Recall from Section II, routing in Kademlia is performed using the XOR metric. Peers with similar first few bits in their IDs are “close” to each other. The resource information can be encoded into the node ID. Instead of the node ID being generated from the IP address only, the node ID is generated using both the resource information and node ID. However, for resource discovery to be possible, the node IDs have to be of some structure, such as peers with similar resources are close to each other in terms of node ID space. The information representing the resources can be prepended to the original node ID, as follows:

$$NodeID = Bits\ assigned\ for\ attributes + NodeID$$

where $+$ indicates concatenation.

For example, we first decide the resource attributes needed and then assign bits for each attribute. If the two resources under consideration are processing power (CPU) and amount of memory, 3 bits can be assigned for CPU (for a maximum of 8 Ghz) and 5 bits can be assigned for memory (for a maximum of 32 GB). We then concatenate all these bits together and with the original node ID. If the original node ID size is n bits, the new node ID size is $3+5+n = 8+n$ bits.

$$NodeID = 3bitsCPU + 5bitsformemory + NodeID$$

This new node ID is used to represent and locate peers. The routing table and routing mechanism stay the same. Each ID is now 8 bits bigger and the overhead is only 8, which is small compared to the original ID size of 160 bits. When a user requests for a node with a list of required resources, the user first calculates the first 8-bits, which indicate the resources needed. The peers which match these first 8-bits are located, and the remainder of the routing process is performed using the original Kademlia algorithm. Peers with the same first 8-bits are closer together and they all have the same available resources.

The main problem with this solution is that it matters which resource is prepended first. In the example above, if a user only requires a certain amount of memory, many query lookups are needed to discover all possible peers. Moreover, the size is fixed. If a user wants to share more than 32 GB of memory, the change is not incrementally deployable and the whole system needs to be updated. Finally, the IDs are not sorted. For example, an ID with 3 Ghz of CPU and 1 GB of RAM is located after an ID with 2 Ghz of CPU and 5 GB of RAM, in terms of XOR metric.

B. Proposed Solution

The proposed solution for resource discovery in a distributed cloud is to use a new concept of local multi-valued hash table. The main idea is to use two different hash tables: one for finding the node location, and one for resource discovery in the distributed cloud. Every node will have two node IDs: one to determine the resource location and another to determine the attributes of a resource. The Kademlia protocol will be used to find the node location; the first DHT behaves exactly as described in Section II. In the proposed approach, the k-buckets in the routing table are used to store the routing information. Each k-bucket's contact contains the following information: IP address, port number, and Kademlia node ID, known as *nodeID1*. Each node will also store a set of local multi-valued hash values of all the nodes in a k-bucket with their IDs generated from the resource attributes. Every node therefore has a Kademlia routing table and a set of local multi-valued node IDs, known as *nodeID2*. Each node will store the mapping information <*nodeID2, set of nodeID1*>, where nodeID1 and nodeID2 are generated as follows

$$nodeID1 = hash(IPAddress)$$

$$nodeID2 = hash(CPU, memory)$$

where $hash$ is a one-way function.

Resource discovery in this model is done by routing to the nearest nodes and finding all the possible nodes which satisfy the user requirement of resources needed. For example, if the two attributes of CPU and memory, nodeID2 is calculated by $hash(CPU, Memory)$ and the values for nodeID2 will be a set of nodeID1. Thus, nodeID2 maps to many different nodeID1. Peers with similar resources are stored under the same key nodeID2. The nodeID2 determines available resources and the nodeID1 is used for routing.

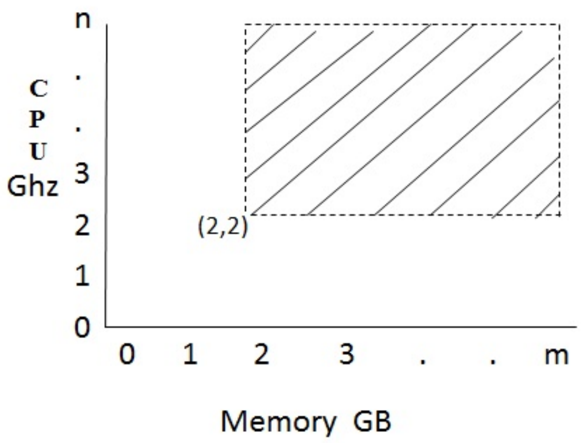
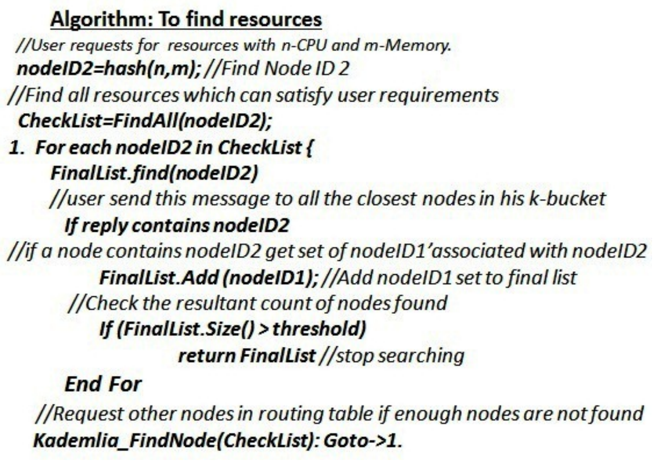


Figure 3. A user requests 2 Ghz of CPU and 2 GB of memory; all peers with resources of at least (CPU, Memory) = (2, 2) need to be discovered. These peers are in the shaded area.

All the attributes can be seen in a n-dimensional space. If 2 attributes are considered, they can be plotted using a bilinear interpolation or represented as a matrix. Figure 3 represents a peer trying to discovering peers with at least 2 Ghz of CPU and at least 2 GB of memory. This is represented by the location (2,2). When a user requests for a resource with 2 GB of memory and 2 GHz CPU, from the figure, all the peers in the shaded region have (CPU, memory)$\geq$ $\{2GHz, 2GB\}$ and can satisfy the user requirements. The algorithm in Figure 4 is used to find all these nodes from the shaded region which can satisfy user requirements.



```
    Algorithm: To find resources
  //User requests for resources with n-CPU and m-Memory.
  nodeID2=hash(n,m); //Find Node ID 2
 //Find all resources which can satisfy user requirements
  CheckList=FindAll(nodeID2);
 1. For each nodeID2 in CheckList {
      FinalList.find(nodeID2)
      //user send this message to all the closest nodes in his k-bucket
        If reply contains nodeID2
 //if a node contains nodeID2 get set of nodeID1'associated with nodeID2
           FinalList.Add (nodeID1); //Add nodeID1 set to final list
       //Check the resultant count of nodes found
           If (FinalList.Size() > threshold)
                return FinalList //stop searching
     End For
   //Request other nodes in routing table if enough nodes are not found
   Kademlia_FindNode(CheckList): Goto->1.
```

Figure 4. Resource discovery model using multi-valued hash tables.

Using the distributed cloud, users can request for a single resource or set of resources. There will be N resource providers which are the nodes in the peer-to-peer network. Currently, the two major attributes for resource discovery are CPU and memory. When a user requests for a resource with n CPU and m memory, the scheme creates a list of nodeID2 that satisfies user requirements. Each of the ID in this list has CPU and memory greater than or equal to $n + t$ and $m + t$, where t is a threshold indicating how much additional CPU and memory are acceptable for efficient allocation. Then nodeID2 is used to obtain a list of nodeID1. The user then searches his routing table for nodeID2. If nodeID2 is found, the user obtains a set of peers with IDs of type nodeID1. If the number of nodes found reaches the threshold level, the user stops querying for more nodes. Otherwise the user sends a request to find nodeID2 to other nodes in its routing table and repeats the process until the number of peers found that meets the requirements, exceeds the threshold.

As an example, a user wants to perform a computation in the distributed cloud that requires 2GHz CPU and 2GB of memory. These are the steps performed to discover all the peers with these requested available resources.

1) Calculate $nodeID2 = hash(2, 2)$.
2) Create set *CheckList*; find all other nodeID2 that can satisfy user requirements, that is (cpu, memory) as (2,2),(2,3),(3,3),(3,2),...,(2+t,2+t).
3) Check the routing table for nodes with initial nodeID2. If found, add corresponding values {nodeID1's} to the set *FinalList*.
4) Check size of *FinalList*. If enough nodes are found, start with resource allocation. Else, repeat the process for other nodes in *CheckList*.
5) If there are not enough nodes that can satisfy user requirements, send a request to find nodeID2 in *CheckList* to nodes in the routing table and repeat the whole process.

Since Kademlia is used, the whole process can be performed concurrently. Moreover since we use a multi-valued hash table to maintain information about attributes corresponding to each node, there is no need to update the original node ID. We need not flood the network with queries to find the node with attributes required as done in range queries for grid services[24]. We see this model works because we insert information regarding nodes and their attributes into multi-valued hash table as we build the routing table and use that information for resource discovery. This way we can use nodeID2 to query instead of long XML queries used by range queries[23,24]. Once the resources are found, resources are allocated based on a game theory model, as discussed in the next section.

IV. Resource Allocation

The proposed model is to use an auction for optimal resource allocation in the distributed cloud. An auction consists of the following: 1) The *resource* to be allocated,

2) an *auctioneer*, who determines the resource allocation; in the model, the auctioneer is the user, and 3) the *bidders*, who provide resources in the distributed cloud; in our model, they are the resource providers. In the game, the *players* are the *bidders* and the *auctioneers*. The main advantage of this type of game is that there are no centralized computations; everything fits within our proposed distributed cloud architecture.

The rules for this auction mechanism are described below. Since the distributed cloud is based on the resources supplied by users, it is free of cost to use. Therefore we need to have a participation factor for each user to avoid the selfish behavior of users. Participation factor shows how much a user has contributed to the system. How the participation factor is calculated is left as future work. In this game, it is assumed that a participation factor can be obtained. The bidders are the resource providers as they try to get incentives for providing the resources and try to increase the participation factor which would be useful when they look for resources.

- Information: The information both *players* knows are about resource description, that is,*CPU, memory, participation factor, latency, and throughput*. The *auctioneer* knows about the latency and throughput from the measurements made by the *bidders*.
- Bids: *Bidders* submit their bids to the auctioneer. Bids constitute the amount of resources the bidders are willing to provide along with the incentives they need for providing them.
- Allocation: Based on the bids the auctioneer calculates the utility function from the information known. Using this utility function, the scheme allocates the user's resources. The utility function shows the value of resources assigned to the user.
- Payments: Since our model of distributed cloud is free of cost, we have a participation factor, δ, assigned to each user, and incentives to encourage users to provide resources.

In this type of auction, the bidder with the highest bid is not necessarily the winner. User i chooses the resources based on the bids and allocates these resources to the user rationally. Each user requesting a resource would not receive ideal resources if its participation factor is low. Users need to provide higher incentives initially to raise their participation factor. The user announces the reserve bid $\beta > 0$. The reserve bid is defined as the most economical bid which makes sure that the user gets his resources such that latency is minimized, throughput is maximized, and meets his requirements. The reserve bid is calculated taking into account all the bids with latency and throughput, with participation factor as major contributing factor. The bids which are closer to β wins. Typically each provider submits the bids to maximize their participation factor. The reserve bid is calculated based on the type of resources requested and their attributes.

A bidding profile is a vector of player bids $\mathbf{b} = \{b_1, b_2, ..., b_I\}$. The bidding profile of user i is represented as b_i and bidding profile of the opponent of user i is defined as $b_{-i} = \{b_1, ..., b_{i-1}, b_{i+1}, ...b_I\}$ which implies that $\mathbf{b} = \{b_i; b_{-i}\}$. User i chooses a resource based on the allocation rule where the utility is defined as

$$U_i(b_i; b_{-i}, \delta) = (latency_i, throughput_i, \delta) \approx \beta$$

Game theory is a strategic mechanism of decision making. In this auction game there are two different mechanisms based on the type of the resources requested by the user. Bidding strategies or auction mechanisms are different based on the type of resources requested; for example whether it is for high performance computing or for high throughput computing. The game is described by considering the *homogeneous* behavior initially, that is, assuming resource providers are submitting their bids to one user. After the bidding is completed, the utility is calculated based on the user requirements, calculated participation factor, latency or throughput, depending on the type of computing being performed.

In a homogeneous system, only one user is requesting a resource at a particular point of time. Then the desirable outcome would be a bidding profile for which the utility function would be closer to β and is called the Nash Equilibrium for the game, where no user can unilaterally deviate, that is,

$$U_i(b_i^*; b_{-i}^*, \delta) \geq U_i(b_i; b_{-i}^*, \delta), \forall i \in N, \forall b_i > 0$$

This auction mechanism will therefore consider the type of resources requested to allocate the resources optimally. The users who provide their resources will be receiving some kind of incentives such as computing with higher priority. The incentives would be based on the participation factor.

V. Conclusion and Future Work

We proposed a novel distributed cloud computing framework. Two main issues in the design of a distributed cloud are addressed: discovery of resources and allocation of resources. An efficient resource discovery algorithm is shown, using multi-valued hash tables. Moreover, the resource allocation is optimal based on the Nash equilibrium obtained from the auction game model. The notion of distributed cloud is an emerging one and the implementation and deployment of such a system would have huge societal impacts. A user can potentially harness the computing power of billions of machines.

As future work, as plan on continuing to address these two issues and build on them. Simulation and other experiments need to be performed to show its efficiency compared to existing schemes and its effectiveness in general. The game theory model will also be extended to include heterogeneous peers. The auction mechanism will be simulated to determine whether resources are appropriately allocated and whether the utility of the whole network is optimized. Other issues we plan to look at include data replication [22] and efficient VM migration as nodes may enter and leave at any time. We also intend to look into peer-to-peer system issues, computation privacy and security.

References

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica and Matei Zaharia, *Above the Clouds: A Berkeley View of Cloud Computing*, Technical report EECS-2009-28, UC Berkeley, 2009

[2] Peter Mell and Timothy Grance, *NIST definition of cloud computing*, National Institute of Standards and Technology, January, 2011

[3] Buyya, R. and Chee Shin Yeo and Venugopal, S.,*Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities*, High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on. IEEE, 2008.

[4] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Warfield, A. (2003, October)., *Xen and the art of virtualization.*, In ACM SIGOPS Operating Systems Review (Vol. 37, No. 5, pp. 164-177). ACM.

[5] Endo, P. T., de Almeida Palhares, A. V., Pereira, N. N., Goncalves, G. E., Sadok, D., Kelner, J., Mangs, J. (2011). *Resource allocation for distributed cloud: concepts and research challenges*, Network, IEEE, 25(4), 42-46.

[6] Han, Zhu, Dusit Niyato, Walid Saad, Tamer Başar, and Are Hjørungnes, *Game theory in wireless and communication networks: theory, models, and applications.* Cambridge University Press, 2011.

[7] Wei, Guiyi and Vasilakos, Athanasios V and Zheng, Yao and Xiong, Naixue,*"A game-theoretic method of fair resource allocation for cloud computing services."*, The Journal of Supercomputing 54.2 (2010): 252-269.

[8] Maymounkov, Petar and Mazieres, David, *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*, Peer-to-Peer Systems (2002): 53-65

[9] Stoica, Ion, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, *"Chord: A scalable peer-to-peer lookup service for internet applications."*, ACM SIGCOMM Computer Communication Review 31, no. 4 (2001): 149-160.

[10] Rowstron, Antony I. T. and Druschel, Peter, *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*, Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg,2001.

[11] Fletcher, George and Sheth, Hardik and Börner, Katy, *"Unstructured peer-to-peer networks: Topological properties and search performance."*, Agents and Peer-to-Peer Computing (2005): 14-27.

[12] Chan-Tin, Eric, and Nicholas Hopper, *"Accurate and provably secure latency estimation with treeple."*, Proceedings of ISOC Symposium of Network and Distributed Systems Security (NDSS). 2011.

[13] Fernández-Baca, David,*"Allocating modules to processors in a distributed system."*, Software Engineering, IEEE Transactions on 15.11 (1989): 1427-1436.

[14] Urgaonkar, Rahul, et al.,*"Dynamic resource allocation and power management in virtualized data centers."*, Network Operations and Management Symposium (NOMS), 2010 IEEE. IEEE, 2010.

[15] Anderson, David P., et al.,*"SETI@ home: an experiment in public-resource computing."*, Communications of the ACM 45.11 (2002): 56-61.

[16] Thain, Douglas, Todd Tannenbaum, and Miron Livny, *"Distributed computing in practice: The Condor experience."*, Concurrency and Computation: Practice and Experience 17(2-4) (2005): 323-356.

[17] Attiya, Hagit, and Jennifer Welch, *Distributed computing: fundamentals, simulations and advanced topics.*, Vol. 53. Wiley, 2004.

[18] Church, Kenneth, Albert Greenberg, and James Hamilton. *"On delivering embarrassingly distributed cloud services."*, Hotnets VII 34 (2008).

[19] Kulbak, Yoram, and Danny Bickson. *"The eMule protocol specification." eMule project*, http://sourceforge. net (2005).

[20] VUZE, *http://www.vuze.com/corp/technology.php*

[21] Hausheer, David, and Burkhard Stiller, *"Peermart: The technology for a distributed auction-based market for peer-to-peer services."*, Communications, 2005. ICC 2005. 2005 IEEE International Conference on. Vol. 3. IEEE, 2005.

[22] Martins, Vidal, Esther Pacitti, and Patrick Valduriez. *"Survey of data replication in P2P systems."* 2006.

[23] Ganesan, Prasanna, Beverly Yang, and Hector Garcia-Molina. *"One torus to rule them all: multi-dimensional queries in P2P systems."*, Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004. ACM, 2004.

[24] Andrzejak, Artur, and Zhichen Xu. *"Scalable, efficient range queries for grid information services."*, Peer-to-Peer Computing, 2002.(P2P 2002). Proceedings. Second International Conference on. IEEE, 2002.

[25] Klyne, Graham, Jeremy J. Carroll, and Brian McBride. *"Resource description framework (RDF): Concepts and abstract syntax."* , W3C recommendation 10 (2004).

[26] Teng, Fei, and Frédéric Magoulés. *"A new game theoretical resource allocation algorithm for cloud computing. "* , .Advances in Grid and Pervasive Computing. Springer Berlin Heidelberg, 2010. 321-330..