Auctions at Edge Clouds

Diogo Paulo Dias, n. 93980
diogopdias@tecnico.ulisboa.pt

Instituto Superior Técnico - Universidade de Lisboa
1049-001 Lisbon, Portugal

Abstract. The evolution of volunteer computing into community networks and community network clouds has taken advantage of lower cost, and higher energy efficiency of lower-end compute and storage devices that populate the edges of internet. This has made the internet's edge richer and filled with a large amount of still under utilized resources.
While users are initially willing to contribute, sustainability of these community edge clouds depends heavily on the users being able to access relevant, interesting services (often deployed as virtualized containers) and obtaining some kind of positive return (incentives) for allowing others to operate on their hardware. The easiest way to incentivize the users to share their own resources is by adapting one or more existing well known trading systems, paying resources or services with currency.
While promoting decentralized planning and scheduling, this work will explore how market and auction based algorithms can be applied to allocate resources and schedule work on edge clouds, advancing the state of the art container orchestrators such as those on Docker.

Keywords: Edge Computing · Volunteer Computing · Market Algorithms · Auction Algorithms · Docker.

Table of Contents

1 Introduction

Cloud Computing is a mature environment heavily used because of its properties like global access, pay for what you use, resource elasticity, and others [31]. To meet these alluring properties, typically Cloud Computing is executed with a couple of geo-centered server farms at the Internet's backbone. This causes Cloud Computing to work mostly at a long distance from the Internet's edge with Wide Area Network latencies and expensive bandwidth for data to reach it.

The Internet Of Things paradigm is rising to new levels, enabling new concepts such as smart cities. A smart city is one which uses information and communication technologies to make the city services and monitoring more aware, interactive and effective [18]. This information feeds the network's edge with a lot of data and this data needs to be transfered to the Cloud data centers in order to be processed. The transfer of large amounts of data will cause bandwidth saturation and a big latency. An effective pre-processing mechanism at the network's edge would reduce the amount of data to send (to be processed or stored at Cloud) reducing the bandwidth consumption and the latency to transport that data. The computing power and storage are also growing at the networks edge: Raspberry PI [9], [29], laptops, desktops, routers, hubs and others. Most of the time these resources are actually underutilized. This inefficiency is opening the doors to new studies, tools and migrations to the Edge Computing in order to provide services with low latency and low bandwidth requirements.

1.1 Motivation

There are still many difficulties in handling edge cloud resources. A structured view of the existing resources, their characteristics and the role of all entities involved in the edge cloud environment is vital. In addition, reconfigurations often need to be performed in order to modify or allocate existing virtual resources, depending on the usage or the service level agreement. Inefficient allocation of resources has a detrimental impact on performance and costs and also impact on the usability of the system.

Market-based resource allocation. Developing resource management techniques that guarantee scalability, performance, manageability and adaptability for the edge cloud environment is crucial to resolve the aforementioned challenges. Traditional approaches, such as system optimization, focus solely on system performance metrics rather than economic factors, such as revenue, cost, income, and profit [24]. Comparing with the system optimization approach, economic approaches and pricing can provide the following advantages:

1. The demand for resources depends on the needs of the users. Also, the resources provided depend on the capacity and needs of the providers. There may be times when the demand is higher than the supply or vice-versa, the supply is higher than demand. Pricing/economic strategies can be used to solve the problem of scarce or abundant resources originated from dynamic demand and supply prices.
2. There are various entities, e.g. stakeholders, end-users, cloud providers, in edge cloud environment that have different objectives, e.g. cost, profit, revenue, income, utility, performance, scalability, as well as different constraints, e.g., the budget and the technology. There are times when these objectives often clash with each other, and this conflict can be efficienlty overcome with an economic/price model. Using economic/pricing models for negotiation mechanisms can result in optimal solutions for entities with different objectivies, achievied in a mostly decentralized manner.

3. In edge cloud environments, the resource providers' profit must be maximized while fulfilling the client requirements. For this reason, price models based on cost minimization and benefit maximization may be used.
4. One of the most important services in the cloud is Video on Demand. This is a service which offers video for people to watch, e.g., Netflix, HBO, Youtube. These providers offer tons of terabytes of media, overwhelming the networks' bandwidth. Price/economic approaches, e.g. smart data pricing, have been used to regulate user demands and have an efficiently use of bandwidth.

Therefore, economic and pricing approaches for resource management have been researched, developed and sucessfully adopted to manage cloud computing deployments.

Cloud Containerization. In order to promote the use of multiple technologies and the deployment of multiple technologies in a heterogeneous environment, it is necessary to have virtualization, isolation, and security. The most used tools to promote virtualization are System Virtual Machines, managed by hypervisors (e.g., ESXi, Xen, QEMU, ...), or containers (e.g., Docker[1], Warden Container, OpenVZ, LXC containers), which are much more lightweight. In Table 1 we show a comparison between Virtual Machines and Containers in several quality attributes.

Parameter	Virtual Machines	Containers
Guest OS	Each VM runs in its virtual hardware and Kernel is loaded into its memory region	All the guests share the same OS and Kernel. Kernel is loaded into physical memory.
Communication	Will be through Ethernet Devices	Standard IPC mechanisms like Signals, pipes, sockets, etc...
Performance	Virtual Machines suffers from a small overhead due to the translation of guest OS instructions to host OS instruction	Containers provide near native performance
Startup time	Virtual Machines take a few minutes	Containers take a few seconds
Storage	Virtual Machines have more storage as the whole OS and the programs that are associated to run	Containers have lower storage consumption due to the base OS being shared

Table 1. Virtual Machine and Container feature comparison, based on [33]

Due to the advantages of Containers (mainly performance, startup, and storage), their popularity is increasing and started being used at large scales in cloud couply deployments.

1.2 Shortcomings of current solutions

There is a lack of decentralized edge-based cloud solutions. Most of cloud solutions: OpenStack[2], OpenNebula[3], Swarm[4] and others, have centralized architectures and operate only in a controlled environment (not using volunteer solutions). The use of

[1] https://www.docker.com
[2] https://www.openstack.org/
[3] https://opennebula.org/
[4] https://docs.docker.com/engine/swarm/

market algorithms to allocate resources in a volunteer environment has been studied, and techniques to improve fairness, utility and truthful price have been applied, but there is a lack of auction-based solutions in a peer-to-peer environment [34].

1.3 Document Roadmap

The rest of the document is organized as follows: in Section 2 we present the main goals of our work and what we want to achieve. In Section 3 it is presented the study and analysis of related work. In Section 4 we propose our solution to address the shortcomings mentioned before. In Section 5 we describe the metrics and techniques used to evaluate our solution; in Section 6 we wrap up all the important information and present some concluding marks.

2 Goals

Our main goal is to develop an Edge Cloud system that enables us to deploy applications in the other edge nodes (normally volunteered resources) driven by a Market-based approach, and using Docker as container technology. The strategy used to assign a machine/resources to an application will be based on market algorithms, more specifically auction mechanisms. The goals needed to achieve in our project are the following:

- Investigate the Edge Cloud environment taxonomy, and different purposes.
- Explore how market- and auction-based algorithms can be applied to distribute work, the different mechanisms, and their advantages and disadvantages.
- The sub-goals of our solution are:
  - Create a peer-to-peer/decentralized system in order to have better scalability, load balancing and avoid single point of failures.
  - Use docker technology to deploy applications due to its lightweight property.
  - Apply an auction algorithm that maximizes utility and price fairness.
  - Test and evaluate our solution with experimental results.

3 Related Work

In this section we will discuss Edge Clouds in Section 3.1, describing its taxonomy, what aspects differ across various environments of Edge Cloud and in the end we classify some tools based on the taxonomy we proposed. The second theme we explore is the different Market Models in Section 3.2. There are many different market models, and each one has is own advantages and disadvantages and differ depending on the usage context.

3.1 Edge Clouds

In this section we will present the main design dimensions of an Edge Cloud environment and the different values/types for those dimensions. These dimensions are **Resource Ownership**, **Architecture**, **Service Level**, **Target Application** and **Access Technology**. The taxonomy can be seen in Figure 1.

Resource Ownership is the dimension which represents the owner type of the resources. Resource Ownership can be divided in three types: Single Owner, Volunteer and Hybrid.

In Single Owner, the devices used to support an Edge Cloud environment are owned by a single entity. Normally these entities already have a group of specific

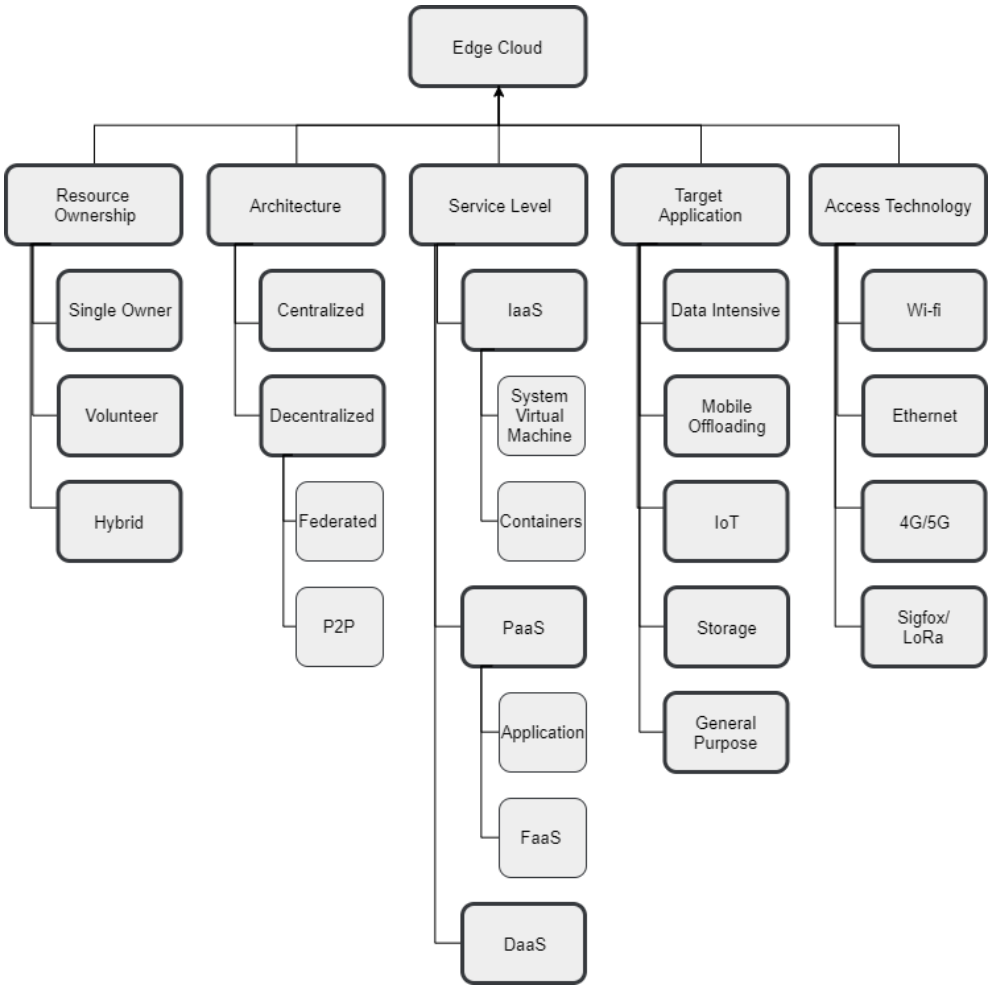

Fig. 1. Edge Cloud Taxonomy

devices and already are configured for the purpose of sharing their own resources to create an edge network. This type of ownership is normally used in big companies which implement their own edge computation infrastructure. They have more control of the system, having more security, reliability and other quality attributes, due to: controlling all the devices and making their own configurations, controlling what types of work will be computed, and the protocol used to communicate between the devices and others.

The Volunteer type differs from the previous one. The resources are shared by end-users and these resources are normally their personal devices like computers, tablets, mobile phones, and others (e.g. Costa et al. [14], Cloudlets [37], Satyanarayanan et al. [21], Cloud@Home [38], Babaoglu et al. [28] and Mayer et al. [32]). One of the biggest differences in this type is that the devices aren't owned by a single entity. Due to that, when we choose a Service Level we don't control the devices used to provide that service, being more prone to disclosure of confidential information or other attacks. This type of ownership may have more end-devices if almost every person shares the resources of their own devices. But these devices (normally personal devices) could have lower capabilities resources in contrast with Single Owner because

the purpose in Single Owner is to share devices, so, they normally buy devices with high requirements.

The last one is a Hybrid type and is a mix of the last two. In this environment, some resources belong to a single entity while others belong to users who share their resources voluntarily. This type of environment with single and volunteer ownership normally happens when the volunteer ownership comes from personal devices like computers, mobile phones, and the single ownership comes from Internet Service Provider devices like routers and hubs (e.g. Nebula [2], Chang et al. [17] and Mohan et al. [26]). This type brings much more power resources to the Edge Cloud but suffers from the disadvantage that some resources may be owned by some malicious users. Volunteer Ownership also suffers from this disadvantage.

Architecture is another relevant design decision dimension. This dimension can be divided in two main types: Centralized and Decentralized. When we are speaking about Architecture we are speaking about the managers/orcherstrators distributed architecture.

Centralized architectures have dedicated nodes to manage and orchestrate all the resources of the Edge Clouds. One problem that arises with this approach is the bottleneck which can be created by the large number of messages between the managers and the worker devices. Also, it suffers from Single Point of Failure issues, i.e., if the managers crash the work stagnates.

Decentralized architectures are the opposite, as all nodes are equal and there isn't a privileged group of managers or orchestrators. This type of architecture tends to scale out better for having a better distribution than the previous one, reducing the bottlenecks and it doesn't suffer from a single point of failure. Decentralized architectures type is divided in two categories: *Federated* and *P2P*. In Federated we have autonomous small clouds (also designated zones in some articles [6]) which provide services. These small clouds can group with other autonomous small clouds to supply a greater/powerful service. Similar, a Peer to Peer architecture makes all nodes equal in terms of responsibilities/work, even if some fails the cloud continues operating (e.g. Babaoglu et al. [28]), and they interconnect themselves creating one Edge Cloud environment.

Service Level is a dimension that refers to the type of service which is provided by a service in an Edge Cloud environment. Similar to Cloud Computing, an Edge Cloud infrastructure offers similar services and for those services different levels. The three main types are Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Data-as-a-Service (DaaS).

IaaS provides infrastructure with CPU, RAM and network capabilities. As infrastructure, we have two different categories: *System Virtual Machines* and *Containers*. In the Virtual System Machine, the devices run a Virtual Machine like VMWare or Oracle Virtual Box, and on top of that runs the operating system and applications supplied by the client. The System Virtual Machine enables to run the clients' applications in a sandbox environment, and also, the client can deploy multiple applications, which can have a relation between them, in one instance of a virtual machine. Using Container technology, the clients only need to provide the application code and their dependencies. Containers are a newer technology that is emerging and is being preferred over System Virtual Machines because they package the application's code and that image is more lightweight than the full system image used to deploy in a System Virtual Machine.

PaaS already supplies the runtime platform (Common Language Runtime, Java Virtual Machine, etc) and the client only needs to send the application code to run remotely. This code differs depending on what PaaS level (which are two) we are using. If it is *Application*, the user must send all the application's code and its dependencies to be deployed (.jar, .exe, .dll) as a self-contained program. The scalability is managed

by the PaaS's provider. The other type is Function-as-a-Service (*Faas*) [22], [3], a new service with rising popularity, and is also named *serverless architecture*. In this type of service, the client only needs to implement some functions to deploy his own application. These functions only contain business logic and are stateless. All the data is saved persistently in databases or blobs and all the logic to receive requests and the code to scale out/in is handled by the FaaS's provider.

DaaS is a service where the provider is offering data storage to the end-users. In contrast with the other types of services, this service doesn't involve computation, the user only needs to provide the data that needs to be stored remotely. The data that is being stored can be files [7] or structured data [8]. Also, different types of data services offers different qualities. These qualities are based on CAP[5] theorem, where we can only choose two of these three quality attributes: consistency, availability and partitioning.

Target Application refers to the type of application we want to deploy on the Edge Cloud. These applications take advantage of some characteristics of Edge Clouds, like low latency. We have five types of relevant applications which could use Edge Clouds: *Data Intensive*, *Mobile Offloading*, *IoT*, *Data Storage* and *General Purpose*.

Data Intensive applications are applications that process large chunks of data. These processes may be cleaning, transformation to other data type/models or aggregation. The chunks of data can't all be processed in one device, so, they are divided into multiple end-devices to distribute the work, augmenting performance. By distributing the data chunks, the network will suffer a higher bandwidth demand which will also cause latency by the processing of moving tons of data. To reduce these disadvantages, data intensive applications exploit the geo-location [2] of the data sources themselves and try to process the data on the closest nodes to avoid the overhead and cost of transfering the data.

Mobile Offloading applications are primarily used by mobile devices or devices which have lower hardware specifications than those required to compute some work. These lower specifications result from the trade-offs in small and mobile devices. To do heavy work in these environments, mobile phones offload their work to end-devices residing on an Edge Cloud infrastructure to make computations on behalf of them. This preserves the battery of mobile phones and the computation is much faster due to better hardware specifications. The Cloudlet work [37] attempts to offer a transparent method for offloading mobile application components to Edge Clouds so as to use the apparently vast amount of resources.

IoT applications are similar to data intensive applications but in this case, while the messages have a smaller payload, the number of messages is much higher. IoT devices try to have a long lifetime to reduce the maintenance and to do that, they take into account the battery drain on daily use. To reduce the battery consumption they send, periodically or eventually, small messages. But a city can have tons of IoT devices, resulting a large number of messages, burdening the network. In order to reduce this burden, some applications aggregate the data in the network's edge before reaching the Cloud [26].

Storage applications [20], [7] differ from the previous ones because they don't have computation. Their purpose is only to store and retrieve data. Using the millions of devices of an Edge Cloud environment they can replicate the data having high availability and reduce the consistency. One downside is also the data may be inspected when stored in malicious' users end-devices. If non-disclosure information is necessary, then encryption is required. To have fast retrieval, data's location can be used to find the nearest nodes with the required data.

[5] `https://en.wikipedia.org/wiki/CAP_theorem`

General Purpose applications are the last all-encompassing type. This type covers all the other applications that didn't fit the previous types. These types of applications can be simulators, servers, and others.

Access Technology is the dimension referring to the technology used to access edge cloud services. One way is by using Ethernet protocol, if the service is near and a physical connection can be used. Another similar approach, but without any physical connection, is by using Wi-Fi (and variantes), more practical to the end-users (being wireless) but can suffer more attacks from spoofing. Mobile devices can also use 4G/5G communication, for example, to offload work to edge cloud end-devices. IoT devices can also use Wi-Fi, but this type of protocol has a high battery consumption. To get high efficiency in the communications other communications protocols like SigFox[6]/LoRa[7] which target IoT devices specifications.

Of the design choices of the work In Table 2 we have a summary. This classification is based on the taxonomy presented in Figure 1. Fields with value '-' mean no information was found or given about that entry. Because of the frequent lack of specific information about Access Technology (relying on higher level protocols such as TCP/UDP over IP), this dimension was removed from the table.

System/Tool	Resource Ownership	Architecture	Service Level	Target Application
Edge-Fog Cloud [26]	Hybrid	P2P	-	IoT
Chang et al. [17]	Hybrid	Federated	Containers	General Purpose
Cloudlets [37]	Volunteer	Federated	Application	Mobile Offloading
Cloud@Home [38]	Volunteer	Centralized	SVM	General Purpose
Satyanarayanan et al. [21]	Volunteer	Federated	SVM	Mobile Offloading
Mayer et al. [32]	Volunteer	P2P	Application	General Purpose
Nebula [2]	Hybrid	Centralized	FaaS	Data Intensive
Babaoglu et al. [28]	Volunteer	P2P	SVM	General Purpose
Samsara [20]	Volunteer	P2P	DaaS	Storage
Past [7]	Volunteer	P2P	DaaS	Storage

Table 2. Edge clouds tools dimensions classification

3.2 Market Models

Market models have been used to solve many issues in cloud environments, solving some challenges and having the advantages mentioned in Section 1. Different market models offer different characteristics and provide different specifications. To choose the market model to use in one service it is first necessary to make a study about the domain where that application will exist and the consumers of that service.

In this section, we will study different market models, their characteristics, and their advantages and disadvantages. We divided the market models into two categories, based on how the prices are set: Provider-Based Pricing and Auction-Based Pricing.

3.2.1 Provider-Based Pricing In Provider-Based Pricing, the prices are set by the resource provider. We address three types of provider-Based Pricing, e.g., cost-based pricing, differential pricing, and profit maximization, where the prices are set by the provider but the approach to find these prices differs from each other.

Cost-based pricing: This technique is a popular pricing strategy to calculate a resources' price based on calculating the resources' total cost and adding, as a desired

benefit, a value e.g. percentage of the cost or constant value. The objective of this strategy is for the resources' provider, ensure some revenue or, in the worst case, guarantee the minimum price to be equal to the cost of providing the resources. The total-cost is created from the fixed cost and the variable cost. The fixed cost is the cost that does not change depending on the supply or the number of requests. These are normally hardware costs, e.g. RAM's price, CPU's price, disk storage's price. In contrast, the variable cost varies with the number of requests or the service provider, e.g. bandwidth used, the disk used, number of servers, and energy.

One of the advantages of the cost-based pricing is the easy way of setting the price, being one function of the internal cost, e.g. the cost to generate the service [10]. One of the disadvantages is that the price doesn't take into account the price of other providers (provider A may offer the same service with the lowest pricing, getting more sells) or the value which the users are willing to pay. Another disadvantage may be the precision to calculate the variable cost. It is necessary to have good metrics and a good monitor to convert those metrics to a cost. This technique has been mostly used to calculate the service cost in geo-distributed data centers [1], [35].

Differential pricing: the previous market model, cost-based pricing, doesn't take into account the value users are willing to pay. This misuse of information reduces substantially the market area of users we want to target. To maximize the profit of providers, it is necessary to attract these types of users. Therefore, it is necessary to know the requirements of these users, know how much they are willing to pay, and find a good price for them.

This technique is called differential price because the provider, may offer resources at different prices to different users, depending on the information aforementioned. The user surplus here is the difference between the overall amount of money users are willing to pay and the total amount of money they pay.

This strategy brings a great profit for the provider can be considered unfair to the users, e.g. one user may pay a much higher value than another for almost similar services. One of the examples of differential prices in the cloud market is the Alibaba[8] group, which offers a discount to use their services, but they need to pay for one year. Another application of differential pricing is used in bandwidth location for dynamic demands in data center networks [13].

Profit maximization: The profit maximization is the usual technique used to maximize the number of resources shared and the corresponding price to get the highest profit possible to the provider. To apply that technique we assume we have the number of resources shared, denominated Q, and the price to share each resource, denominated P. The profit of a provider is given by the formula, $\pi = R(Q,P) - C(Q)$, where the π is the profit, the R is the revenue of one provider based on a number of resources shared and the price to share that resources, and the C is the cost to operate and share the resources. The cost could be divided into fixed cost and variable cost (these costs were explained in the Cost-based pricing technique).

We want to find the optimal Q* where the profit is maximized $Q^* = \max(\pi)$. With the optimal Q* we find the optimal price, to share our resources, based on demand curve. The demand curve is a linear function and represents how much the clients are willing to pay for a resource based on its quantity. The demand curve is represented by the following expression: $P = a - bQ$. The constant a and b are proper parameters. If we find the optimal Q* using the demand curve we find the optimal price $P^* = a - bQ^*$. To find the optimal Q*, we first compute the revenue function and the cost function. Using these functions we obtain the profit function, we just need to subtract the revenue function from the cost function. After obtaining the profit function, we find the Q* where the profit was highest. Then, we use this Q* value and obtain the

[8] https://www.alibabacloud.com/

optimal price (P*) by intercepting the Q* value (which is a straight line) with the demand curve.

These iterations of steps can be seen in Figure 2. One downside it's the price that doesn't take into account the market competition. One difficulty in this approach is to determine the demand curve. To create this curve, different techniques are applied, some use randomness, others follow a distribution function like gaussian.

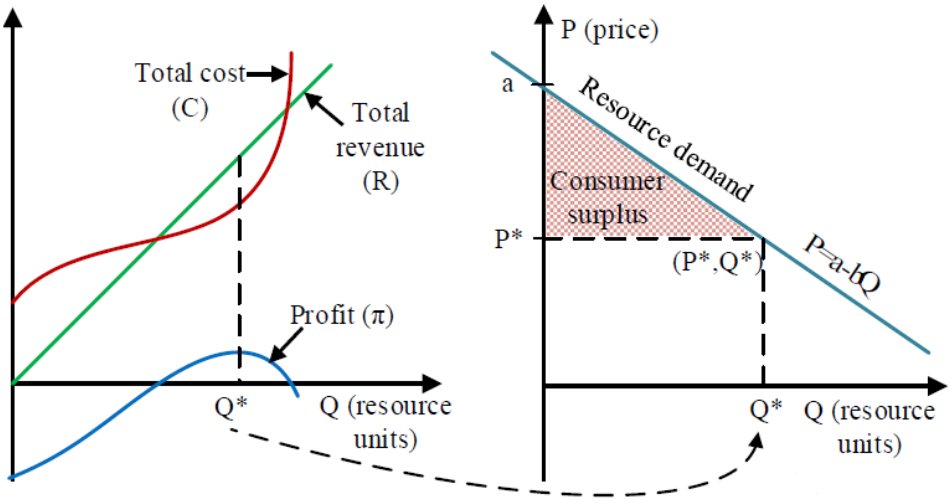

Fig. 2. Find the optimal price based on profit maximization, extracted from [25]

3.2.2 Auctions

Auctions are one very relevant type of Market model. An Auction, in its simplest form, is a process where one item is being bid by a group of buyers and the item is sold to the highest bid [16]. This is one of the processes of the auction but there are others with different actions and we will speak about it bellow. In an auction, we may have up to four types of peoples involved in a transaction. The *bidder* is the person who wants to buy the product and *bids* to obtain it, e.g., end-users, cloud tenants. The *seller* is the person who offered the product to be sold to obtain some profit, e.g., cloud provider. Sometimes the *seller* is the same person as the *auctioneer*. The *auctioneer* is normally an intermediary agent that conducts the auction and ensures a winner is decided. In this section we will speak about the different dimensions an auction algorithm it can have. These dimensions are **Confidentiality**, **Direction**, **Unity** and **Symmetry**. The taxonomy structure can be seen in Figure 3.

Confidentiality: this dimension refers to if the price that the bidder's bid, is known by the other participants or not. It is open-cry if the bids are known by all participants. If the bidders don't know the bids of the other participants are called sealed-bid.

Direction: this type of dimension focus on where the competition comes from. It is a forward auction if the bidders fight among themselves for getting the item. To do that the winner must bid the highest value. It is called reverse auction, when we only

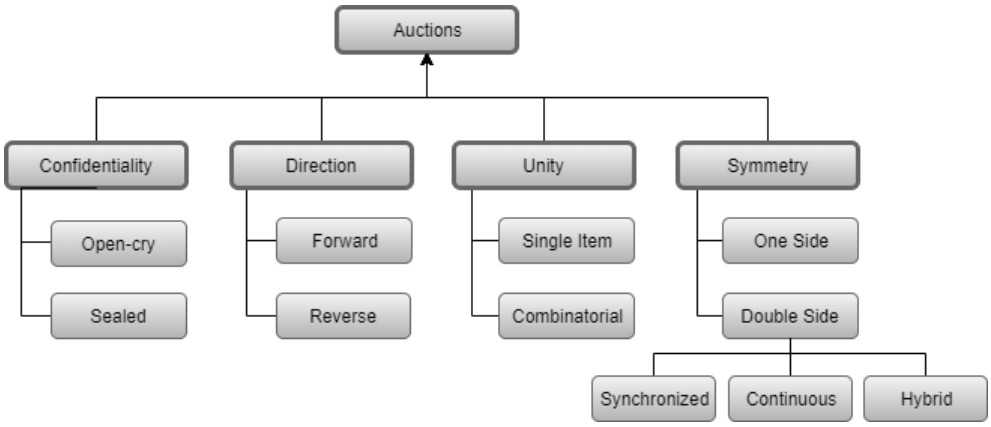

Fig. 3. Auctions Taxonomy

have one buyer, and the sellers, compete among themselves, by lowering the price of their product, until the buyer choose the item with the lowest price.

Unity: this dimension classifies whether the good being bought is a solo unit, or a combination of items. In a single item the customers only bid for one item, for combinatorial, they can make combinations of goods and buy them as a bundle.

Symmetry: this dimension is related to whether the competition is only among the buyers or between the buyers and the sellers. It is referred to as one-sided if the competition is only between the bidders or the sellers (never both at the same time). It is called double-sided if the competition is between bidders and sellers, meaning, the buyers submit *bids* and the sellers can submit *asks*.

There are many of one-sided auctions. Some of them are: *English* auction, *Dutch* auction, *first-price sealed-bid* auction and *second-price sealed-bid* auction (the second-price sealed-bid auction can also be named Vickrey auction).

In an *English* auction, the auctioneer starts to sell an item from a low price (normally this low price is protected to ensure some return to the seller) and the buyers bid the item by ascending the price. When no more buyers bid the item, the last bidder or the person who bid a higher value wins the item. It is also open-cry, forward auction and can be a combinatorial or single item.

A *Dutch* auction is also open-cry, can be combinatorial or single item and also follows a forward auction, but the price to buy the item is descending. The auctioneers start selling the item from a high value (normally higher than the real value), and at each time reduces the value until some buyer bids the item to buy. Normally this type of auction is faster than the English auction because there is no possibility of recursing competition of bids between buyers; the first one to accept the price, wins the resource.

The *first-price sealed-bin* differs from the previous ones, in the the confidentiality dimension. The first-price sealed-bid is sealed, meaning, bidders don't know the value of others' bids. Each bidder secretly bids an item. After all the buyers bidding, the seller/auctioneer, orders the bids based on the price bid, and the buyer with the highest bid wins the item, paying the value bided. There only exists one bid per buyer.

In *second-price sealed-bid*, the process is the same as the first-price sealed-bin, the difference is that the winner of the auction, does not pay the value of his bid, but pays the value of the second highest bid.

In double-side auctions, the buyer bids and the seller asks, creating more competition compared with one-sided auctions, and much more fairness due to both par-

ticipants, the seller and the buyer, participate actively in the auction, generating a demand and supply profiles. The winner of this type of auction depends on two different aspects: aggregation and resource divisibility. If aggregation is not allowed, for each ask, only one bid can be assigned. Resource divisibility means whether the resource can be divided among multiple buyers. Gode and Sunder [12] further divide double auction into three categories: *synchronized* double auction (or *discrete-time* double auction), *continuous* double auction (*CDA*) and *semi-continuous* double auction (or *hybrid* double auction). In a continuous double auction, a buy order or sell order can be submitted at any time, and if there is a match between the buyer and the seller, the trade can be made at that exact moment. In contrast, in synchronized double auction, traders move at the same time.

In combinatorial auctions. The users may want to bid multiple resources like CPU time, memory, and network bandwidth. The buyers normally, in this case, bid a bundle that contains multiple resources/goods. The advantage of this approach is that, in order to obtain a group of resources, the bidder only needs to attain to one auction, in contrast to participating in multiple auctions, in each one, obtaining just one type of resources. The disadvantage is the difficulty to obtain the correct price for multiple combinations of resources/goods, then it is hard to find the set of bids that maximizes the revenue generated. This problem is considered a NP-Hard problem [23].

3.3 Relevant Systems

In this section we will speak about systems who proposed different auctions algorithms (often based on one-sided or double-sided but with some nuances) or use other type of market mechanisms.

3.3.1 Envy-Free Auction Mechanism The mechanism proposed by Bahreini et al. [36] handles allocation of resource available at two levels of the system by combining two auctions, position and combinatorial auctions. The position auction disables the use of resources from different levels (levels represent where the resources are located, in edge or cloud), also it is used to characterize the user's preference (edge or cloud). The combinatorial enables to bid a bundle of resources in contrast to bid n times for n different/equal resources.

3.3.2 Multi-Round-Sealed Sequential Combinatorial Auction A multi-round-sealed sequential combinatorial auction mechanism is proposed by Zhang et al. [15]. This auction is combinatorial, meaning the good can be a bundle of items, also it is multi-round. In one-round auctions, all the bidders submit their bids at the same time, and the auctioneers choose the winners and match them with the resources. This approach was not used because the architecture proposed by Zhang uses multi-service providers, and one-round auctions need one controller following a centralized architecture.

The auction mechanism is thus divided into three stages: *bid strategy*, *winner determination*, and *payment rule*. At each round, the users send their resources requirements and bids to all auctioneers (service providers) (bid submission). After that, the service provider chooses the winner based on who brings the highest utility to the service provider. The utility is based on the bidding value provided by the bidders. The bidders who fail to obtain one service provider are moved to the loser vector. The bidders from the loser vector will bid again in the next iteration/round receiving a bid improvement. The iterations finish when the number of max rounds has been achieved or the difference between the utility of the current round with the previous

round is lesser than a threshold. In the last stage, payment rule, the bidders who won the auctions pay the value equal to the second-highest bid, following the sealed second-price auction (Vickrey Auction) approach.

3.3.3 Tycoon Tycoon [19] is a distributed market-based resource allocation on an Action Share scheduling algorithm. In their architecture, the authors separated the mechanism from the strategy. The strategy interprets users' and applications' specifications and the resources desired. One example of that is: one web server may have more concern about latency than throughput and is, therefore, willing to consume lesser resources, but that resources should be located near the clients. The mechanism provides incentives for users truthfully value the resources, and the providers provide good resources. Their Auction Share is similar to proportional share, but enables them to specify how they trade-off throughput, latency, and risk.

In **proportional share**, a group of buyers offers some value to buy a group of products that are divisible. Then the products will be divided into the buyers based on their bids. The percentage of products provided to each buyer is proportional to the bid value in comparison to other bid's value. This mechanism maximizes the allocation of the product to a buyer because everything will be assigned to one buyer. In Figure 4 we can observe three end-users bidding for a memory RAM of 100GB. After each one bid his own price, the auctioneer divides the RAM proportional based on the bid's value. Proportional share has various variants, another type of use is, instead of dividing resources to multiple buyers, the buyers always obtain the total resources, and what they are buying is the time of CPU usage, each one obtains the CPU proportionally to the bidding price.

One of the advantages of Proportional Share is that it offers a higher time of utilization and lower time of reservation.

Two drawbacks are that the product must be divisible for a group of bidders to bid, the product will be divided, based on the bidder's value, to all the bidders, and the Quality-of-Service is not secured. The Quality-of-Service is not secured because one may want to rent all the products or none, but using the Proportional Share, if another buyer bids too, one will lose some products to that bidder, ruining one goal of getting all or none.

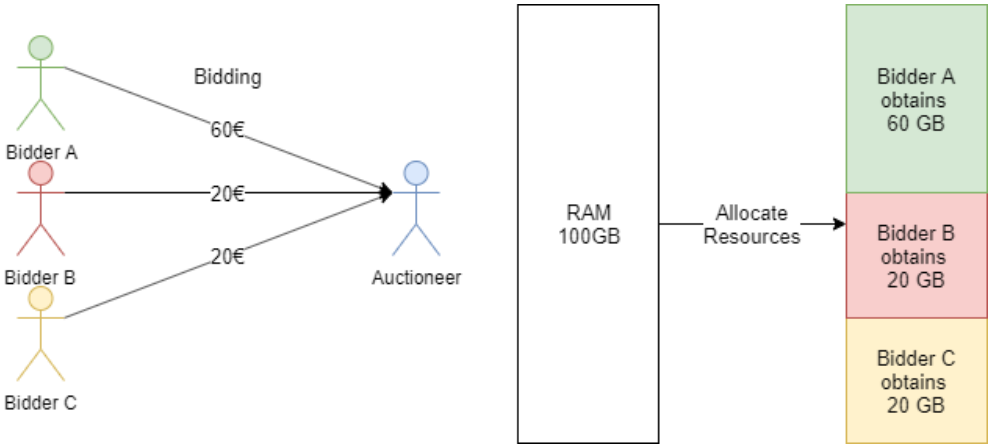

Fig. 4. Proportional share example

For fine-grained resources, it is used as a first-price sealed-bid auction or the second-price sealed-bid auction. The bidder with the highest bid wins allocation to a window slice time processor (each buyer gets their respective time to run their applications on device's cpu). Because of being distributed, the system is fault-tolerant and allocates resources with low latency.

3.3.4 PeerMart PeerMart [11] is a distributed technology which enables trading of services using Double Auction algorithms over a peer to peer network. Their goal was to maximize the consumers' utility and find sellers offering a particular service at a low price, and the providers goal is to offer their services at the highest price possible and maximize their profit. The intermediary peers are responsible to match consumers with the providers respecting the consumers' and providers' requirements efficiently.

The authors chose using pricing mechanisms to incentivize peers to provide services, like file storage. By choosing a peer to peer architecture, they don't have a central authority to maintain the prices bid by the bidders or the sellers. One way to communicate could be using broadcasts, but this does not scale and doesn't guarantee that all peers are reached. They proposed to maintain routing tables at intermediary peers. Then peers use these routing tables to find the peer who offered a given service for a given price. The use of double auctions is derived from single-sided auctions and it has the disadvantage of being consumer- or provider-oriented. One of the problems of implementing a peer to peer infrastructure is that malicious or faulty users may exist. To solve this problem, PeerMart uses public cryptographic keys to identify the sender.

The auction algorithm works as follow: A provider (consumer) who wants to provide its service (consume a specific service), sends an offer (request) to the respective broker, which is composed of a set of peers. The broker replies with the highest buy price (lowest sale price) offered by another peer. After the provider (consumer) receives the information, it sends a bid to the broker applying its own strategy. Then the broker receives the bid and with that, chooses one of the two options: After receiving the offered price, there is no match if the offered price is higher (lower) than the current bid price (ask price). Therefore, the offered price is either dropped or stored on the table for future use. If there is a match, the price is sent to the peer who has the highest bid. The price paid to the provider by the buyer, is the mean between their offered prices.

To implement a peer-to-peer network, they choose to use FreePastry[9]. It is a tool that is implemented in Java and the the overlay of the network is based on Pastry.

3.3.5 Bellagio Bellagio [5] is a resource management tool that allocates resources using combinatorial auctions. To discover the resource existing in the network they use SWORD [27]. The users submit the bids to a centralized auction, and they use XOR language as the bidding language in order to simplify the learning curve of the end-users. The auction is periodic and receives requests for heterogeneous resources like disk space, memory and bandwidth.

3.3.6 Lin et al. Lin et al. [39] propose a dynamic auction mechanism to allocate resources in a edge computing environment. They made two contributions: i) the introduction of peak/off-peak concepts into the resource allocation, ii) the system contains two types of tasks, background, and float. The first contribution enables the cloud provider to increase efficiency and its revenue in a varying demand environment.

[9] https://www.freepastry.org/

The second contribution enables the devices to distribute the resources to end-users and have its own background process. The revenue is obtained from the inputs to the background task and also the resources shared with the users. They use second-price sealed bid, each user bids to a cloud service provider. The cloud service provider collects the bids and orders them. They find how much capacity they can provide, e.g. k, and from this capacity, they say the price is the (k + 1) highest bid. The k highest bidders obtain the resources with the price from the (k + 1) highest bid. They employ a truth-telling method due to the price to pay being determined by their own bids.

3.3.7 Double Multi-Attribute Auction Wang et al. [41] proposed a resource allocation model based on the Double Multi-Attribute Auction (DMAA). Their model focus on three important steps. Firstly they transform the non-price attributes in a Quality Index that represents the assesment to the previous transactions. After that, they use Support Vector Machines to predict the price. Lastly, they use Mean-Variance Optimization to obtain an efficient solution to allocate the resources (choose the winners) to different users.

Their system is divided in three actors: the Cloud Resource Provider (CRP), Cloud Resource Consumer (CRC), and Auction Organizer (AO). The CRP provides the resource in exchange of a payment. The CRC pays to a CRP to allocate resources. The AO is the auction organizer responsible to collect the bids and asks, match the transactions, and select the winners.

To calculate the price submited by the CRP/CRC a group of steps are necessary. In CRP, they obtain three non-price attributes, namely, Quality of Service (QoS), Level of Delivery (LoD) and Level of Spiteful Quote (LoSQ). The CRC also follows the same logic but doesn't have the QoS attribute. Then they use these attributes and transform them in a Quality Index by using a neural networks algorithm. The activation function used was the Sigmoid function.

After having the quaility index, they use Support Vector Machines to predic the price. In order to find the estimated transaction price, they also use quality index and other metrics (created by themselves) like reserve price of provider, ration supply demand and expected sale amount of provider. After that, the CRP/CRC obtains the estimated transaction price. To train the Support Vector Machine classifier, they use historical samples from the previous auctions and input information.

Then, the AO makes the match between the CRC and CRP based on this information, and determines the winner by using Mean-Variance Optimization. This algorithm enables to find the most efficient way to distribute the resource to the users.

3.3.8 Auction Based Resource Co-Allocation Auction Based Resource Co-Allocation (ABRA) [4] is a model that improves upon a previous novel combinatorial auction model called multi-unit nondiscriminatory combinatorial auction (MUNCA). The new model penalizes the non-allocated resources after an auction, having a better resource utilization and hence increasing the revenue. Their model can be mathematically formulated by using integer linear programming. In the paper, it was also proposed a set of five new heuristic algorithms that are based on well-known meta-heuristic techniques. These five heuristics are: (i) simulated annealing, (ii) threshold accepting, (iii) list based threshold accepting, (iv) variable neighborhood search and (v) genetic algorithm.

3.3.9 Reverse Batch Matching Auction Wang et al. [40] proposed a Reverse Batch Matching Auction (RBMA), that is based on reverse auction, but has additional features like batch matching, to improve the reverse auction efficiency, and

twice-punishment mechanism to prevent fraud and milicious users. RBMA has three participants in the system: Cloud Resource Consumer (CRC), AI (Auction Intermediary) and Cloud Resource Provider (CRP). AI the is key component to control the system. It stores the resource information, applies the reverse auction, batch-matching, and twice-punishment mechanism. The CRC and CRP send their tendencies/intent (e.g. bids, resource ammounts) to the AI. Then, the AI starts the auction process that is divided in three stages: waiting period, preparation period and auction period. The waiting period is the period that the buyers and the sellers send their tendencies/intent to the AI, and these are ordered in ascending order, if are seller bids, and in desdending order, if are buyers bids. The start of the auction is marked when the system receives the first buyer's parameters. In the preparation period, it is where the AI selects the buyers and sellers that can participate in the auction. This selection is a time-restriction. Auction period is the last stage. There is a matching between the CRC and CRP based on the bidding prices. At the end of the auction it is used the twice-punishment mechanism. After that, the resource allocation mechanism uses Immune Evolutionary Algorithm and the transaction price to otimize the resources allocation. Also, the CRP service is graded to improve future services that CRC obtains. To evaluate the auction, three evaluation criteria were applied by them: market efficiency, user satisfaction and quality service.

3.3.10 Combinatorial Double Auction Resource Allocation Samini et al. [30] proposed a Combinatorial Double Auction Resource Allocation (CDARA). To simulate the prototype of this auction, it was used CloudSim, which is a Java-based simulator for simulate cloud environments in order to extract metrics and evaluate the efficiency of the auction algorithm. In their environment, there are four entities: the user, the broker, the cloud provider, and the cloud market place. The cloud market place is composed of cloud information service (CIS) and auctioneer. From the article, it stems the cloud market place as being a centralized entity. CDARA is divided into seven communication phases.

At first phase, the cloud providers send their resources, and their respective prices, to the CIS. The users send their tasks to the broker and, for each task, the broker gets the list of resources that match the requirements to run that task.

The second phase, the broker generates bundles (a combination of resources) and the price for each bundle. The cloud provider does the same action. Both send price (bids) to the auctioneer.

In the third phase the auctioneer communicates to the broker and the cloud providers the end of the auction.

In the fourth phase, the winner is determined. In this phase, the users and cloud providers are ordered depending on what resources they are bidding/sharing and the respective price.

In the fifth phase (called resource allocation), the auctioneer checks if the cloud provider has the necessary requirements, requirements defined by the user, to run the tasks. If the first cloud provider cannot fulfill the requirements, the auctioneer passes to the second cloud provider. After the requeriments of the first user are satisfied, the auctioneer applies the same procedure for the next user.

In the sixth phase it is selected which pricing model to use to decide the payable price by a user to a cloud provider for allocating resources. To use this model, it is used the number of requested items by the user and the number of offered items by the cloud provider.

In the last phase, the user sends the task to run in the cloud provider's resources. And the user makes the payment to the cloud provider.

3.4 Analysis and Discussions

After studying and classifying taxonomically the edge cloud environment, in our solution, it will be followed a distributed nature, more precisely, we will aim to implement a peer-to-peer architecture. The reason of this choice is because, peer-to-peer is much more scalable and our solution will target all end-devices that users use, which could be millions of devices. Of course, this choice also brings disadvantages, and one of them is the existence of malice users.

On top of our overlay network, it will be used an auction mechanism to match the resources being sold to a buyer. The auctions follow more a price/demand curve comparing with the other market mechanisms because multiple sellers or providers influence the price

It was hard to choose which type of auction we will be based on to implement. This difficulty was due to the different characteristics they have between them, and these differences aimed to solve different problems. We decided to adapt auction mechanisms, that would fit better in our domain (sharing resources) and our architecture (peer-to-peer).

4 Solution Proposal

In this section, we present our proposed solution with architecture, mechanisms and platform, which enables the allocation of resources by deploying applications on other devices. In order to have scalability and not suffer from a single point of failure, the solution is based on peer-to-peer (all nodes are equal) architecture, and it will include auctions algorithms to assign a resource to an application. To deploy the application on remote devices, container technology will be used, more precisely Docker, due to its lightweight characteristics. We can observe in Figure 5 our proposed solution.

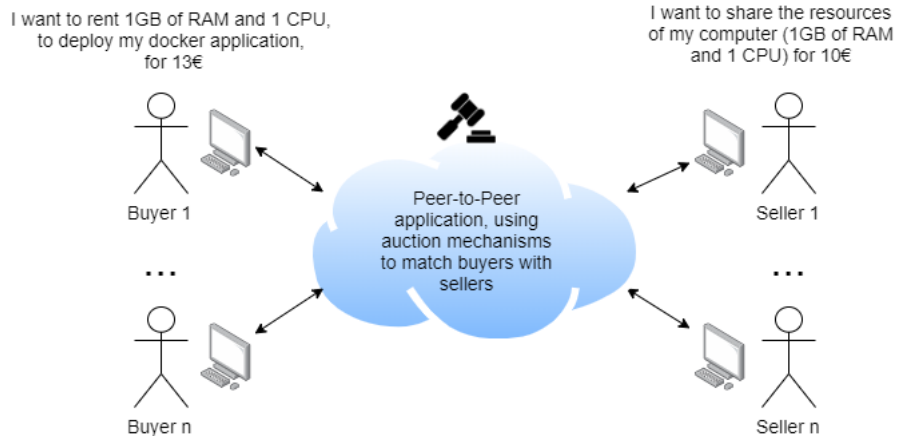

Fig. 5. Solution Overview

Our approach is to develop a peer-to-peer architecture (with client application), that enables to allocate the resources of other peers by using a number of selected auctions, more specifically: one-sided, double-sided and combinatorial auctions. Combinatorial enables to the user to buy multiple resources in only one bid, and because it is double-sided, the sellers and buyers participate actively to determine the price of a group of resources. The auction algorithms are based on existent ones [30], [11],

[39], but they will be extended and adapted to our distributed architecture and our domain. We will focus first to extend and adapt the double auction algorithm [11] and then the others.

To facilitate the sharing of resources, it is necessary to have an identifier (Global Unique Identifier). This identifier needs to contain the information of CPU and memory RAM capacity that a user is selling or buying. This identifier, it will help to match the buyer to an auction that is selling the same resource.

In order to implement this approach, two issues need to be addressed in particular. The first one is the auction/resource discovery. Because it is a peer-to-peer network, it is important to have an efficient search of auctions. When a client wants to deploy its own application, it wants to find an auction that matches the requirements (e.g. a client may want at least 8GB of memory) as fast as possible. So, it's imperative to have an efficient auction/resource discovery. The other problem is to schedule the resources using auctions. It is important to have a fast auction and also reliable, in order to assign a resource to a buyer, and both the seller and the buyer be satisfied with the outcome.

4.1 Distributed Architecture

As aforementioned, it will be implemented a peer-to-peer architecture. Using this type of architecture, a lot of challenges arise. One of them is the information not being centralized. As a consequence, we need to have a way of tracking/routing information. In our context, we want to buy/sell resources, and we want to know who are the nodes that are selling/buying the resources that we want. One way could be broadcasting a message to all nodes, which would not scale due to the overhead on the networks. Another way is by having a structured overlay network, sending the messages to the respective nodes that have the information that we want.

Auction Discovery (Overlay Structure). A structured peer-to-peer architecture is a network overlay organized in a specific topology, and the protocol ensures an efficient search/routing even with millions of nodes. The most common type of structured peer-to-peer architecture is to implement a distributed hash table, based on consistent hashing. To implement an efficient peer-to-peer architecture, we found two approaches to our solution.

In the first approach, each user, independently of being a buyer or a provider, will get a Global Unique Identifier (GUID). This GUID is created based on their intention to buy or sell, and what type of resources want to obtain. In Figure 6 we can see the composition of the GUID, each circle represents a bit, and it is only used 8 bits to simplify the explanation (in reality, we can use 64 or 128 bits). The first section, Intention Section, shows if the user is with the intent to buy or to sell. The Random Section is to mitigate collisions of GUIDs for two users that are selling or buying the same resources, in that way, they will have different GUIDs even if they sell the same thing. The last section, Resources Requirements Section, represents the resources that the user wants to buy or sell, for example, if a user wants to buy 1GB of memory and 1 core, this could be encoded to 0111. This is an example where the request of 1GB of memory and one CPU could be mapped to the code 0111, the fourth bit (counting from right to left) represents the number of CPUs: bit '0' represents 1 CPU and bit '1' represents two CPUs. The other three bits represent the memory, where '111' represents 1 GB of RAM which, in this example, could be the maximum of memory that is possible to share.

This allows fast lookup and routing of requests and resources and balance the load uniformly across nodes/providers. In this type of structure, we want to find the user that is selling or buying a given resource. These resources are associated with an

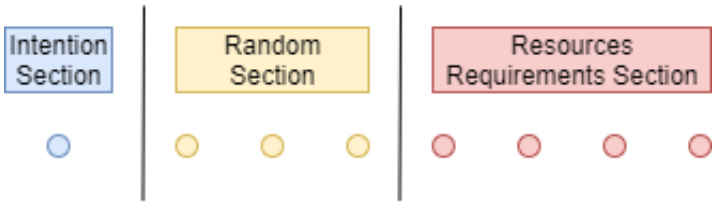

Fig. 6. GUID decomposition from the first approach

auction mechanism in order to be sold. For example, imagine that we only have 6 bits to encode all GUIDs in the ring and the resource we are trading is a memory that varies between 0 and 1GB. If I want to sell 0.5 GB of Memory, I join the ring with the GUID 4 and try to send messages that I am selling 0.5GB to the buyers that want to buy 0.5 GB. If I don't find any suitor, instead of finding one buyer that wants to buy 0.5GB of memory, I try to find two buyers that want to buy 0.25GB of memory. The other way around also works, when a buyer wants to buy 0.5GB of memory, he could send a message to the peer with GUID 4, supposing the 0 came out in the random bits section, with the intent to buy those resources.

The advantage of this approach is that, they are able to know the nodes' intent, whether it is selling or buying, based on GUID. We could instantly find the user's GUID that is selling/buying what we want and talk to him to trade. Reducing the number of messages to find an auction that is selling a given resource and reducing data replication, or don't need to tell the other peers that I am selling a given resource because my GUID already is showing my intent.

One disadvantage is that a buyer could be buying resources more expensive than the necessary. For example, we have two sellers selling 1GB, one at 10€the other at 20€. Both sellers will have different GUID because of the random bits section. When a buyer calculates a GUID to find a seller that is selling 1GB, it could be talking with the seller that sells for 20€, depending on the number the buyer obtained from the random bits section. Paying 10€more that the necessary because there was another seller that was selling the same resource but a much cheaper price.

In the second approach the users' GUID is random, e.g., the hash of its own IP. After obtaining its own GUID, depending on the GUID, some nodes will aggregate the auctions of other users. As we can see in Figure 7, node A, with GUID 4, is responsible to aggregate all requests to sell or buy from resources that are encoded from 1 to 4 (low number of bits were used to simplify the explanation). The node with GUID 8 is responsible from 5 to 8, and so on. If there aren't nodes with aggregators' GUIDs, the successor of that nodes will be responsible for aggregate.

For example, we already have a node with GUID 4, and this node is responsible to aggregate all intentions to sell and to buy from 0 to 0.5GB. After that, a seller enters the ring with GUID 6 (it was randomly obtained), and it wants to sell 0.5GB, so, sends that information to node with GUID 4. The node with GUID 4 receives that information and saves it. Afterward, a buyer (node 1) enters in the ring and wants to buy 0.5GB, so asks for sellers to node 4, because it knows it is responsible for aggregate/save all sales from 0 to 0.5 GB. Finally, node 4 responds to node 1 that node 6 is selling what it requires.

The advantage of this approach is that one node only needs one GUID, in contrast with the previous one, each node could have more than one GUID depending on the number of resources to buy or sell.

One disadvantage of this solution is the propagation of information. When a user wants to sell some resources, it notifies the responsible nodes to save that information. Having much memory consumption than the previous solution.

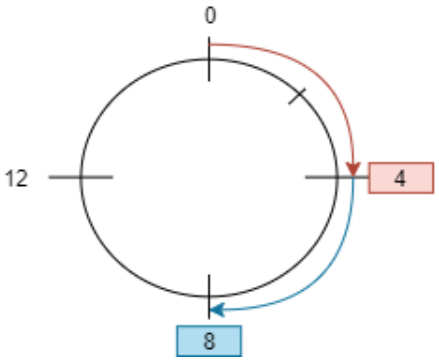

Fig. 7. Ring structure aggregation

To implement our distributed architecture, we will first use the second approach, then we will try to implement the first in order to evaluate the tradeoffs between them.

4.2 Resource Scheduling (Bidding Algorithm)

After finding the required resources, it is necessary to match those resources to a user. As it was previously mentioned, it will be used auction algorithms to assign resources to a user. The auction algorithms are inspired on (and extended) some auction algorithms that were already created. To have a better study we will choose three different types of auction algorithms: combinatorial double auction model [30], distributed double auction [11] and dynamic auction mechanism [39]. These three auction mechanisms fall into three different categories (one-sided, double-sided and combinatorial), having a better diversity in the study in order to have better evaluation and improvement. Thus, those algorithms by themselves are not suited for auctioning cloud resources in edge cloud (either because they don't handle resources, or they don't scale in a peer-to-peer deployment), so, they will be adapted and extended to our domain of the problem and our distributed architecture. As it was previous mentioned, we will focus first to extend and adapt the double auction algorithm [11] and then the others. This choice is due to, the double auction mechanism enables fast assignment of a buyer to a seller and both participate in the price.

Bids Lifetime. Due to the auctions nature, and the limited resources, it is necessary to control the bid's validity. The bids' TTL (time-to-live) vary between three possibilities: *time-window*, *count* and *immediately*.

Supposing that we are using an English auction, we need to reserve a time-window to wait for bids. The more time we wait, the higher is the probability to have more bids. Another approach is, instead of using a time window to receive the bids, we could use a counter. Configure a number maximum of bids for each auction, and only after that number is reached, we choose the winner. The problem with these approaches is the delay to match a buyer to a seller. These approaches may be used in combinatorial double auction [30] or dynamic auction mechanism [39]. It is also possible to combine the time-window with the count threshold, only waiting for one of them to finish.

The last way would be doing a match immediately. This last time dimension only works with double auctions, due to the sellers also participate in the auction, and both (the buyer and the seller) show their intent. The immediately match is used in distributed double auction [11].

Prototypical Example. We will explain one use case to sell resources and to deploy an application in those resources. In this example, it will be used the second approach to implement a peer-to-peer network and the auction algorithm will be based on PeerMart [11]. In this type of network overlay, we have groups of nodes responsible to aggregate the biddings and asks from the buyers and sellers. We need to distinguish the biddings from the asks, their values, the resources that which one wants to obtain, and the identifier of the responsible node that made the request. In Table 3 we can observe the information stored in a aggregator node to enable transactions in a peer-to-peer environment. Due to the restrictions of memory, we can't save all the offers from buyers and sellers that could be unbounded over time. To address this constraint, two techniques may be used. One is, distribute the information to other nodes in order to use all the existent resources. The other option is to remove some offers no longer relevant for decision, for example, remove the lowest bids for the same resource, or remove the highest asks for the same resource.

Bid value	Resources	Buyer GUID	Ask value	Resources	Seller GUID
9	124653247	6546836	10	124653247	9995354
6	124653247	6574396	13	124653247	7658070
30	847558735	7428993	10	657984368	9995354

Table 3. Data stored in aggregator nodes

For example, suppose we want to allocate a node with 2 cores and 1 GB of RAM. First, we obtain the sellers that are selling those resources and the respective price. Based on that price we make a bid to the node. Because we are using a double auction [11], if the bidding price is higher or equal to the lowest ask from the sellers, a match is found and a transaction is started (lines 2-6). In the transaction phase, the buyer pays to the seller, and also deploys its own docker application in the seller's resources. If the bid is lower, it is saved on the node in case new sellers enter on the ring and sell their resources at a lower price (line 7). We can observe in Algorithm 1 the behavior, in pseudo-code, after an aggregate node receives a bid from a buyer:

Algorithm 1 Node behavior after receiving a bid

```
1:  function BIDDING(buyerGUID, buyerBid, resources)
2:      asks ← askTable.filter(a− > a.resources == resources)
3:      descendingOrderAsks ← asks.orderByAscendingPrice()
4:      lowestAsk ← descendingOrderAsks.getFirst()
5:      lowestAskPrice ← lowestAsk.price
6:      if lowestAskPrice ≤ buyerBid then
            askTable.Remove(lowestAsk)
            return lowestAsk
7:      else
            bidTable.Add(buyerGUID, buyerBid, resources)
8:      end if
9:      return "BidSaved"
10: end function
```

The behavior is similar when the aggregate node receives an ask from a seller. The difference is that it searches the highest bid from the bid table, and if no match was found, the ask is saved in ask table.

4.3 Software Architecture

Based on the behavior explained in the previous Sections, we now present the modules that contain those functionalities. For each module, we explain their responsibilities and interactions. A model view type using decomposition and uses styles can be seen in Figure 8.

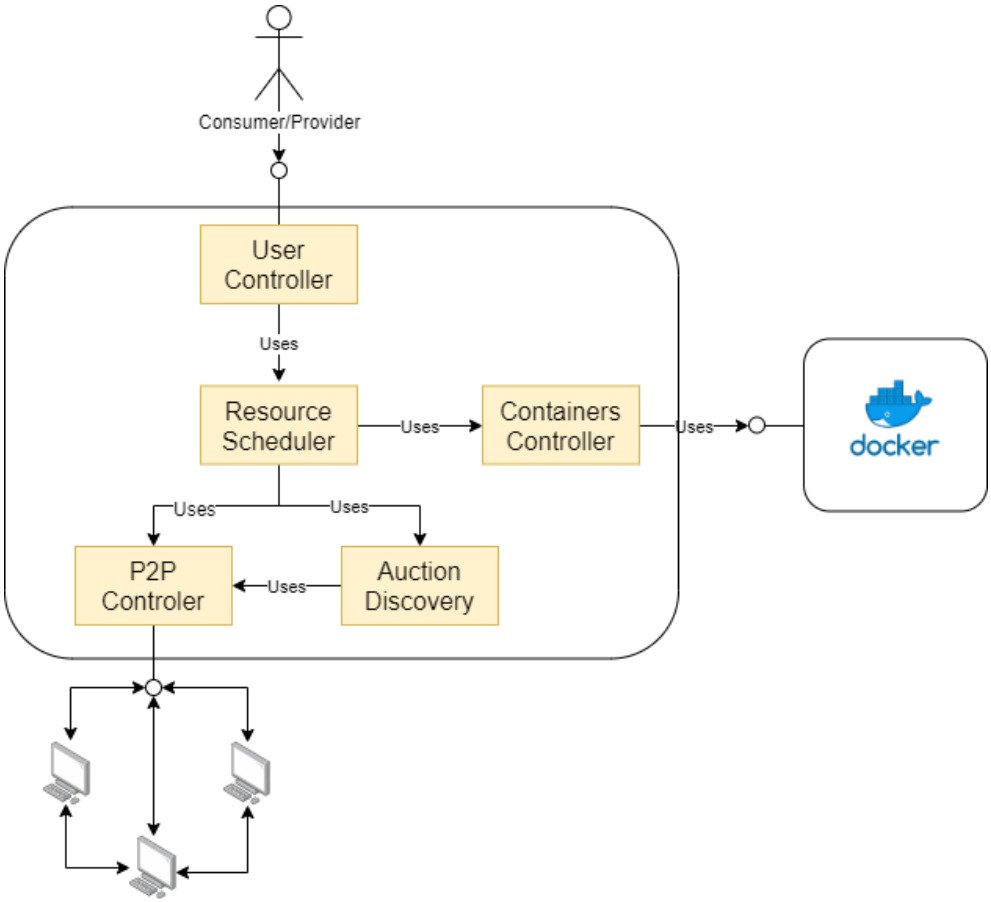

Fig. 8. Software architecture

User Controller: this is the module responsible to trigger the users' requests. It offers a bridge between the user commands and the actions made by the application.

Containers Controller: this is the module responsible for managing the containers deployed in the machine. After the resources are assigned to a user, this controller is used to deploy the application in a container to run.

Auction Discovery: this is the module responsible to find auctions that sell resources required by the user.

Resource Scheduler: this is the module responsible to deploy the users' application is other machines using Auction Discovery module, or to offer its own resources to deploy a users' application.

P2P Controller: is the module responsible to communicate with other peers and creates the structured overlay network.

5 Evaluation Methodology

Below, we enumerate the metrics that we will use to evaluate our prototype solution:

1. Allocation Success Rate: This metric helps to assess how effective is our approach to find auctions and allocate resources. This is important in a peer-to-peer environment, where we don't have a centralized component and some allocation requests could not be fulfilled, causing retries (overhead) thus will compromise the level of ineffectiveness in our prototype.
2. Overhead Extra Memory Space Distributed: this metric assesses how much memory is consumed by a node just by running our program. It is necessary to know if it is sustainable to a node and if all the information is evenly or approximately distributed between nodes. Also, it allows to verify how much this metric grows when the number of nodes increases.
3. Overhead Extra CPU/Network: this metric helps to calculate the overhead of the device by running our prototype. This helps to know if our application is CPU-intensive or IO-intensive even when the user is idle (without buying or selling anything, the prototype is just running in the background).
4. Scalability: this metric assesses how much our solution is degraded by accommodating a large number of nodes. This degradation could be due to the exponential increase in the number of messages by adding nodes.
5. Reliability in a Crash Environment: this metric helps to observe if the solution is able to retry the deployment of applications that were interrupted, due to the crash of the device, in another device (even if with longer execution or worse performance).
6. Ideal Price Deviation: This metric will tell how much more the buyers had paid to get a resource that was being sold at a lower price somewhere else in the network. One example is we have two users, selling the same amount of memory (1GB) but one for 10€and another for 5€. Then a buyer arrives and offers 10€to buy 1GB of memory. In an ideal environment, the transaction should happen with the user that is selling for 5€, but because of the absence of centralized control in a peer-to-peer architecture, he could buy from the user that is selling for 10€. This raises a problem where buyers don't buy the cheapest resource because of the distributed environment.
7. Distribution of Revenue: this metric studies the revenue of the different sellers and buyers, in order to check the efficiency of the pricing mechanism. Verify if doesn't exist attackers that explore the mechanism and obtain large quantities of money.

To have a realistic simulation, the user's requests were generated from Google Cluster Data[10] (CPU, RAM). This data is provided from 12.5k-machine cell in May 2011.

6 Conclusion

Our thesis will study the use of auctions mechanisms in an edge environment in order to allow the allocation of resources. With that in mind, we will be implementing a solution that is built on top of a peer-to-peer network overlay which enables the deployment of Docker applications in other computers by buying those resources. The *modus operandi* to assign a resource to a Docker application will be based on auctions mechanisms.

[10] https://github.com/google/cluster-data

Firstly we made an introduction in both cloud and edge computing, secondly, we present the advantages of using market mechanisms to allocate resources. Lastly, we compared Containers and Virtual Machines technology.

After research on edge cloud environment, we presented a taxonomy for edge cloud. It was also explored the market and auction mechanisms used to allocate resources. We also proposed our taxonomy for auction mechanisms. At the end of related work, it was explored some articles that offer different auctions mechanisms to allocate resources.

Afterward, we proposed our solution where we highlight two fundamental problems (auctions discovery and resource scheduling) and how we will aim to solve it. At the end of the section, we exposed our architecture and its components. We ended the document with the methodology used to evaluate and study our work.

References

1. Albert Greenberg, James Hamilton, David A. Maltz, Parveen Patel: The Cost of a Cloud: Research Problems in Data Center Networks. ACM SIGCOMM computer communication review, vol. 39, no. 1, pp. 68–73, Jan. 2008 (2008)
2. Albert Jonathan, Mathew Ryden, Kwangsung Oh, Abhishek Chandra, Jon Weissman: Nebula: Distributed Edge Cloud for Data Intensive Computing (2017)
3. Alex Glikson, Stefan Nastic, Schahram Dustdar: Deviceless Edge Computing: Extending Serverless Computing to the Edge of the Network. Proceedings of the 10th ACM International Systems and Storage Conference Article No. 28 (2017)
4. Ali Haydar Özer, Can Özturan: AN AUCTION BASED MATHEMATICAL MODEL AND HEURISTICS FOR RESOURCE CO-ALLOCATION PROBLEM IN GRIDS AND CLOUDS. Fifth International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control (2009)
5. Alvin AuYoung, Brent N. Chun, Alex C. Snoeren, Amin Vahdat: Resource allocation in federated distributed computing infrastructures. In Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT Infrastructure (2004)
6. Amin M. Khan, Felix Freitag, Luís Rodrigues: Current Trends and Future Directions in Community Edge Clouds (2015)
7. Antony Rowstron, Peter Druschel: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. Proceeding, SOSP 01 Proceedings of the eighteenth ACM symposium on Operating systems principles, Pages 188 - 201 (2001)
8. Avinash Lakshman, Prashant Malik: Cassandra - A Decentralized Structured Storage System. ACM SIGOPS Operating Systems Review, Volume 44 Issue 2, April 2010 (2010)
9. Claus Pahl, Sven Helmer, Lorenzo Miori, Julian Sanin, Brian Lee: An Information Framework for Creating a Smart City Through Internet of Things (2016)
10. Costas Courcoubetis, Richard Weber: Cost-based Pricing. John Wiley & Sons, pp. 161–194 (2003)
11. David Hausheer, Burkhard Stiller: PeerMart: The Technology for a Distributed Auction-based Market for Peer-to-Peer Services. IEEE International Conference on Communications (2005)
12. Dhananjay (Dan) K. Gode, Shyam Sunder: Double auction dynamics: structural effects of non-binding price controls. Journal of Economic Dynamics and Control, Volume 28, Issue 9, July 2004, Pages 1707-1731 (2004)
13. Dinil Mon Divakaran, Mohan Gurusamy, Mathumitha Sellamuthu: Bandwidth allocation with differential pricing for flexibledemands in data center networks. Computer Networks, vol. 73, no. 1, pp. 84–97 (2014)
14. Fernando Costa, Luís Veiga, Paulo Ferreira: Internet-scale support for map-reduce processing. Journal of Internet Services and Applications, vol. 4, no. 1, pp. 1-17 (2013)
15. Heli Zhang, Hossein Badri, Heli Zhang, Fengxian Guo, Hong Ji, Chunsheng Zhu: Combinational Auction-Based Service Provider Selection in Mobile Edge Computing Networks. IEEE Access

16. Hui Wang, Huaglory Tianfield, Quentin Mair: Auction Based Resource Allocation in Cloud Computing. Multiagent and Grid Systems, 2014, Volume 10, Number 1, May 2014, pp. 51-66 (2014)
17. Hyunseok Chang, Adiseshu Hari, Sarit Mukherjee, T.V. Lakshman: Bringing the Cloud to the Edge (2014)
18. Jiong Jin, Jayavardhana Gubbi, Slaven Marusic, Marimuthu Palaniswami: An Information Framework for Creating a Smart City Through Internet of Things (2014)
19. Kevin Lai, Bernardo A. Huberman, Leslie Fine: Tycoon: a Distributed Market-based Resource Allocation System
20. Landon P. Cox, Brian D. Noble: Samsara: Honor Among Thieves in Peer-to-Peer Storage. Proceeding, SOSP 03 Proceedings of the nineteenth ACM symposium on Operating, systems principles, Pages 120-132 (2003)
21. Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, Nigel Davies: The Case for VM-based Cloudlets in Mobile Computing (2009)
22. Michał Król, Ioannis Psaras: NFaaS: named function as a service. Proceedings of the 4th ACM Conference on Information-Centric Networking, Pages 134-144 (2017)
23. Muralidhar V. Narumanchi, José M. Vidal: Algorithms for Distributed Winner Determination In Combinatorial Auctions. Agent-Mediated Electronic Commerce. Designing Trading Agents and Mechanisms pp 43-56 (2005)
24. Nguyen Cong Luong, Ping Wang, Dusit Niyato, Wen Yonggang, Zhu Han: Resource Management in Cloud Networking Using Economic Analysis and Pricing Models: A Survey. IEEE Communications Surveys & Tutorials, Volume: 19, Issue: 2, Secondquarter 2017 (2017)
25. Nguyen Cong Luong, Ping Wang, Dusit Niyato, Wen Yonggang, Zhu Han: Resource Management in Cloud Networking Using Economic Analysis and Pricing Models: A Survey. IEEE Communications Surveys & Tutorials, Volume: 19 , Issue: 2 , Secondquarter 2017 (2017)
26. Nitinder Mohan, Jussi Kangasharju: Edge-Fog Cloud: A Distributed Cloud for Internet of Things Computations (2016)
27. Oppenheimer, D., Albrecht, J., Patterson, D., Vahdat, A.: Scalable wide-area resource discovery. In: USENIX WORLDS. vol. 4 (2004)
28. Ozalp Babaoglu, Moreno Marzolla, Michele Tamburini: Design and Implementation of a P2P Cloud System (2012)
29. Paolo Bellavista, Alessandro Zanni: Feasibility of Fog Computing Deployment based on Docker Containerization over RaspberryPi (2016)
30. Parnia Samimi, Youness Teimouri, Muriati Mukhtar: A combinatorial double auction resource allocation model in cloud computing. Information Sciences, Volume 357, 20 August 2016, Pages 201-216 (2016)
31. Peter Mell, Timothy Grance: The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology (2011)
32. Philip Mayer, Annabelle Klar, Rolf Hennicker, Mariachiara Puviani, Francesco Tiezzi: The Autonomic Cloud: A Vision of Voluntary, Peer-2-Peer Cloud Computing (2013)
33. Rajdeep Dua, A Reddy Raja, Dharmesh Kakadia: Virtualization vs Containerization to support PaaS. IEEE International Conference on Cloud Engineering (2014)
34. Saurabh Garg, Rajkumar Buyya: Market-Oriented Resource Management and Scheduling: A Taxonomy and Survey. Cooperative Networking, Chapter 14 (2011)
35. Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, Harbinder Bhogan: Volley: Automated Data Placement for Geo-Distributed Cloud Services. NSDI, San Jose, California, USA, Sep. 2010, pp. 17–32 (2010)
36. Tayebeh Bahreini, Hossein Badri, Daniel Grosu: An Envy-Free Auction Mechanism for Resource Allocation in Edge Computing Systems. 2018 Third ACM/IEEE Symposium on Edge Computing (2018)
37. Tim Verbelen, Pieter Simoens, Filip De Turck, Bart Dhoedt: Cloudlets: Bringing the cloud to the mobile user (2012)
38. Vincenzo D. Cunsolo, Salvatore Distefano, Antonio Puliafito and Marco Scarpa: Cloud@Home: Bridging the Gap between Volunteer and Cloud Computing (2009)
39. Wei-Yu Lin, Guan-Yu Lin, Hung-Yu Wei: Dynamic Auction Mechanism for Cloud Resource Allocation. Proceeding CCGRID '10 Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing Pages 591-592 (2010)

40. Xingwei Wang, Jiajia Sun, Min Huang, Chuan Wu, Xueyi Wang: A resource auction based allocation mechanism in the cloud computing environment. IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (2012)
41. Xingwei Wang, Xueyi Wang, Cho-li Wang, Keqin Li, Min Huang: Resource Allocation in Cloud Environment: A Model Based on Double Multi-Attribute Auction Mechanism. IEEE 6th International Conference on Cloud Computing Technology and Science (2014)

Planned Gantt Chart