Investigating Pricing and Negotiation Models for Cloud Computing

Raluca Suzana Andra

August 22, 2013

MSc in High Performance Computing
University of Edinburgh
Year of Presentation: 2013

Abstract

Negotiation models have been adopted by all types of businesses today to improve company image. Cloud computing has become used more and more to the point of being a profitable business. The providers have started to charge users for what they use. However, the pricing model available today in cloud computing does not support interactive negotiation. This will help to produce more profit as well as to increase the users' satisfaction because they will have a say in it. This dissertation presents a new negotiation model that has been proven to satisfy both the provider and the consumer. The provider's satisfaction consists in maximizing the profit and utilization, while the consumer's satisfaction is based on the price going to be paid.

Acknowledgements

I want to thank and show gratitude to my supervisor, Mr Neil Chue Hong for the liberty to choose my own theme, for the continuous advise, support and trust in my own capacities to finalize a new negotiation model that was not developed before. I want to thank my classmates for the wonderful moments we had along the year in High Performance Computing. I also want to thank my professors for the passion and devotion shared with us in discovering the secrets of High Performance Computing.

Contents

List of Figures

List of Tables

1 Introduction

We live now in an era where technology is developing very fast. In every business, companies are trying to overtake their rivals by providing a better product or by lowering the prices. This way they will attract more consumers and the business will keep growing. This is important for providers in information technology related resources, where commonly customers have a high degree of choice and the resources are relatively fungible yet non-storable. For example, mobile carriers are always trying to change their plans to attract more customers. In the last couple of years mobile carriers have introduced new pre-paid plans where users get certain benefits monthly without making a long term commitment. This gives users the freedom to change providers in case they are not happy with the plan they have. This is an advantage for the provider because the users will not feel constrained which will make them feel like they have the power. Investigating this, we have concluded that the price and benefits given in a plan are one of the most important delimiting factors for consumers to choose one company, besides reliability.

The same applies to cloud computing. Every cloud provider is trying to obtain the highest market share and one way to do this is by trying to create different pricing schemes like the mobile carriers are doing. Amazon, Microsoft and Google, some of the leading providers, have decreased their charges as part of a continuous competition. In the case of Amazon, their prices have been reduced by 12%, and Microsoft has decreased their prices to match those from Amazon. However, there are still lots of improvements that can be made to the cloud pricing scheme. One of them is adding interactive negotiation to the pricing. Negotiation is becoming an important factor in every business because it forces the providers to lower their prices in order to make the business grow. This will allow the users to advertise the price they are willing to pay for the resources. With time, all the providers will introduce negotiation if it proves to be worth. Then the users can choose whoever they think provides more benefits for them. At present, cloud computing provides similar models of pricing to mobile carriers, except the pre-paid monthly plan with no long term commitment. Later on in the dissertation, this will be analysed to see whether it brings benefits to the cloud or not.

This dissertation will introduce cloud computing and its use of pricing schemes and explain why it is important for a cloud provider to have a pricing model. Section 2 contains a case study on the two leading cloud computing providers pricing schemes, discusses why the pricing models are needed and what models they currently provide. After this, Section 3 includes a new negotiation mechanism that handles interactive negotiation between the consumers and the providers. It also introduces a new approach for pricing. In Section 4, we described the design of the implementation. Section 5

discussed about the implementation of the new negotiation model, while section 6 includes analysis between the new model and the ones already provided by Amazon and Microsoft. The dissertation ends with a conclusion, followed by any future work that can be done in this area.

2 Background

Nowadays, cloud computing has become a profitable business[1] since providers have started charging users for the resources they use by creating different pricing models. This section introduces the notion of cloud computing and how it has developed over the past years. Later on in this section, we will describe the pricing models that are available today as well as what makes a pricing model beneficial. The section will conclude with a comparison between Amazon AWS's and Microsoft Azure's pricing.

2.1 Introducing Cloud Computing and the pricing models

Cloud Computing refers to what will happen if the applications and services will be moved on the internet, respectively on the cloud[3]. It has been evolving very fast over the last couple of years[2]. Nowadays, it provides elasticity as well as security and reliability which was not very well defined when it was first introduced. In the early age, cloud lacked elasticity, reliability and security. With time, cloud computing became more and more popular so providers had to improve their services to attract more users. Cloud Computing provides certain core characteristics to the user:

- Shared Infrastructure - uses a software model that is virtualized allowing sharing of physical resources, storage as well as providing network capabilities,
- Dynamic Provisioning - the cloud provides resources based on the current needs of the consumers. This must ensure reliability and security. Dynamic provisioning is also known as elasticity[1], which used to be an issue in the early ages of Cloud Computing.
- Network Access - resources need to be accessed from anywhere. The Cloud handles all types of applications, including the ones for mobile services.
- Managed Metering - used for monitoring and optimizing the services as well as for billing the consumers.

[1]Elasticity means expansion based on the consumers demands. Allowing a provider to have an infinite set of resources.

Since it is a relatively new technology, cloud computing has been seen as[4]

- The idea of having infinite resources on-demand without users making provisions far ahead;
- Cloud providers do not need to have all the resources up front, they can increase the number of resources when a user requests more;
- Consumers pay for resources on a short-term bases as needed, and they can release instances once the computations are done.

Many companies are providing cloud computing services today. This dissertation only investigates the top two leading providers as identified in [5]

- Amazon - which is the leader in cloud computing. It has offered services to the public since 2006;
- Microsoft - is the closest competitor Amazon has at the moment. It has offered services since 2010.

The cloud provides different service models that are targeted at various types of consumers as presented below[6]:

- Infrastructure as a Service(IaaS) - offers physical or virtual machines, targeted mostly at developers,
- Platform as a Service(PaaS) - provides computational resources which allows applications and services to be developed and hosted on a platform,
- Software as a Service(SaaS) - this is the application side of the cloud where the customers have all the applications pre-installed. In this case the users do not need to go through all the hassle of purchasing a license as long as they are using the pre-installed applications.

Although we have described the different service models offered, we are going to concentrate on the IaaS for the rest of the dissertation since we believe it is the most used. Cloud computing has been expanded into four different deployment types as shown in Figure 2, each defined by its own characteristics which satisfies every users needs:

- Public Cloud - This type of cloud is opened for everyone. Customers can request instances and use as much as they need. (e.g Amazon)
- Private Cloud - This is used internally in a company or organization by the employees that stores all the data inside the cloud.

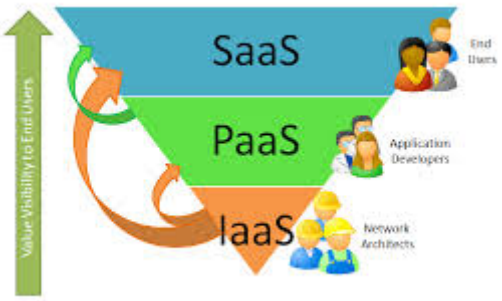

Figure 1: The pictures shows how all three types of service models that cloud computing provides.

- Hybrid Cloud - This is a combination of one or more clouds that remain as their own entity but are tied together.
- Community Cloud - This type of cloud shares the infrastructure between different organizations that have common concerns.

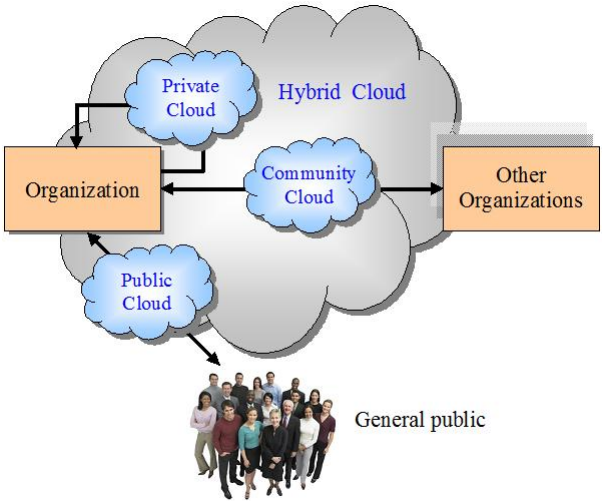

Figure 2: How the four deployment types work and how they differ.

For the rest of the dissertation we will just focus on developing a pricing scheme for the public cloud. However, it can be implemented on all the other types since they all require some sort of pricing models.

People are starting to see cloud computing as the "next wave of information technology". It stays at the base of new business ideas because a lot of big companies

have started using a private cloud inside the company in order to keep all the data. A recent study shows that in US the cloud is already used by more than half of the businesses[7]. The research reflected that almost 54% of the companies are using either public or private cloud while still leaving space for industry growth. Businesses have started to adopt cloud computing because of the benefits[8] that brings them. One thing cloud can offer is flexibility. When starting a project either the requirements, such as network interfaces, volumes might change along the way or they might not be sufficient.

In cloud computing this is not a problem because additional requests can be added later after the servers have been set up. Users can decide that the instance requested does not have sufficient memory in which case they can pay a small price to get more, based on their needs. This is one of the most important reasons companies have started using cloud in the first place. Another important characteristic of the cloud easier disaster recovery. Through disaster recovery, the cloud provider is entitled to resolve all the internal issues relevant for the cloud as well as maintaining and supporting the servers. A study has shown[9] that a company that uses cloud was able to resolve a problem nearly four times faster than a company not using cloud. Cloud computing also saves users a lot of time. The benefits of working on the cloud are that users do not have to do any maintenance since it is handled by the supplier. Cloud computing has no need for capital expenditure since it is a pay-as-you-go service that does not require any start-up cost. It can, however, get expensive when the user requires to do large computations over a long period of time. Another benefit provided is increased collaboration. It makes things easier for employees inside a company to share and sync files or apps with each other wherever they are while receiving real time criticism. In technology, this means that the cloud can be accessed from everywhere in the world. Users can choose to store their data on a cloud server that is different than their location, paying the price for that region. Users today go for the more secure choices because they want to ensure their data is protected. Last but not least, cloud computing has been proved to be environmentally friendly. It helps decrease the energy consumption and carbon emission by 30%[10].

2.2 Cloud Pricing Scheme

Providers also offer different pricing plans for the cloud. Later in this section, we will talk about the pricing plan for the two cloud computing leaders, Amazon and Microsoft. The following pricing plans can be taken into consideration[14]:

- fixed recurring pricing - the users can set up a plan where they can have the same amount of resources every month. This is similar to the contract plans available today in cloud computing.

- variable pricing by resource consumption - in this case the price can vary based on how many resources there are used at a time. This has been adjusted for the on-demand plan where the users are paying based on how many resources they need.
- variable pricing by time - the price varies based on how much time the client uses the resources for. This has been implemented in the spot pricing provided by Amazon.
- cost multipliers - help the providers increase the price by a factor. This has not been implemented in any of the cloud computing pricing schemes. They are mostly used by insurance companies.

Because there is not much information about pricing models on the cloud computing, we are going to discuss them from an economics point of view. Pricing can be seen as a chargeback model which can be based on different businesses. In IT a chargeback model is defined as a user paying for what he/she has used after usage. The model needs to be accurate, auditable, flexible and scalable. In order to develop a model there are some steps that need to be followed: analysing all the costs needed, identifying all the items that need to be billed(in this case the instances provided by the cloud), finding a pricing model by choosing different pricing options that could potentially give benefits to the consumer without making it worse for the provider; identifying, collecting and deploying tools to mediate the data into a billing system[13].

In this dissertation, we only considered the pricing strategy: what strategies are currently available and how new ones can benefit both the provider and the consumer. Clouds computing must provide a *good pricing model* that is beneficial for both parties. It is sometimes hard to find a balance in which both sides agree with the price set. A *good pricing model* is defined as a price that will bring no loss to neither the provider nor the consumer. From the consumer's point of view a better pricing model is one where they will pay a lower price for the resources requested, while from the provider's point of view, they should not go beyond the lowest price that provides 0% profit for them as well as increasing the utilization. The consumers point of view can be summarized as the *user satisfaction*. By a lower price we mean that it will only include energy consumption, cooling, support in case it is done by a third party, and possibly getting additional resources. The pricing model that we work with for the rest of the dissertation is described below:

$$price\_set = energy\_cost * amount + cooling + support + profit \quad (1)$$

where:

- energy_cost is the energy price per hour for a KW. For the purpose of this dissertation the energy price will be 0.07$/KW. The cost for energy is incurred whether or not the machines are in use.
- amount refers to the energy used by a server for an hour,
- cooling refers to the price needed to cool a machine per hour. This is also incurred regardless if the machines are in use or not.
- support relates to the price required to maintain a server. However, not all of the cloud providers have a third party support in which case there is not cost for the support.
- profit is everything extra that the provider can get. This will go for buying extra machine in order to increase the utilization. The profit is increased or decreased based on how much the consumer is willing to pay.

As mentioned above cloud computing works on a pay-as-you-go basis, allowing the user to pay for what they use. It might not be so cheap on the long run but it will definitely save the user a lot of headaches. Providers offer different types of hardware at various prices because they need to maintain those machines. The costs, however, need to be relatively small in order to attract users. No one will be interested in paying lots of money for a long period of time if they do not have any benefits. For this reason the providers want to keep the prices low and attract more users that will demand more and more resources in order for them to develop a business. One way to decrease the price as we can see in the price set above is by lowering the profit.

For this reason, cloud providers should have multiple pricing schemes from which users can choose. This gives them the liberty to pick what they want without being constrained. By providing users with different pricing schemes either short-term or long-term they can decide which one suits their needs best.

Providers do not disclose too much information about their pricing scheme, except price, duration and the instances they provide. The pricing-schemes available today charges users based on per-minute, per-hour, six-months contract, one-year contract, three-year contract, etc. The process is very simple. A user picks a provider from which he/she wants to rent resources, probably based on the cheaper price or reliability. Then the user selects an instance that will suit his/her needs based on the amount of computation required and the complexity of the job. In case the system needs additional storage or networking, the provider allows each user to add them later on. Users need to be careful for how long the instances have been running for. For example, if the user picks Amazon, they will be charged by the hour regardless of the fact that they only used 1 minute or 59 minutes; but if they go with Microsoft Azure they will be charged per minute. This makes it easier for the user because they will not have to check for

the time constantly. In Section 2.5.1, we will analyse if the pricing per minute is more beneficial for the consumer rather than the pricing per hour.

The price has been set low in order to increase utilization. However, the user does not have a say in setting the price. Current models do no provide interactive negotiation[11], however, this is going to be one improvement that can potentially bring benefits for both providers and consumers. Both of them can get to an agreement from which they can both benefit. For example, setting the price that the consumer wants without making the provider lose any money will be satisfactory for both of them. The provider can have some price boundaries for which they can settle. If the user goes for a price in those boundaries then the negotiation can close. However, the user will have no knowledge of what the boundaries are, so it will be based on how much he/she is willing to pay for resources and whether the user is willing to increase the price. Also by interacting, the provider can understand better the users requirements. The pricing models currently available can only handle basic knowledge of negotiation. The provider sets the price for different types of instances which means that the user can choose to either accept them or reject them. In case they reject the price, the users can go to a different provider. The negotiation can also be done for the amount of instances. If the user needs more instances than the provider allows, they can negotiate to come for an agreement. For example, if a user requires 25 medium instances and the provider can only offer 20. Then the provider can give those 20 medium instances to the user and give him/her another 5 large instances. The user will then have to pay the price for 20 medium instances plus the price for 5 large instances in which case it will be more expensive. The negotiation process described here can be good from the provider's point of view because it will increase the utilization and the user will pay more for the other 5 instances, while from the consumer's point of view it will be good because it will satisfy the user's requirements. This is one way of doing negotiation. More about negotiation will follow in section 3. The negotiation mechanism presented in that section provides interaction between users and providers that will come to an agreement regarding the price. Later on, we conduct the analysis to see how the interactive negotiation behaves based on the pricing provided today by the clouds.

Since the dissertation tackles the pricing model for cloud providers let's take a deeper look into how the pricing can be differentiated. We have found out that the pricing scheme can be either static or dynamic. In static pricing, the prices are fixed allowing the user to pay per time interval used, requiring optimization to be done only once. This is seen as a drawback[12]. The user has to pay for the resources used. Resources refer to the different types of data structures active in query execution. There are different query executions used by the cloud to fasten the query. One model of static pricing is the pay-as-you-go model where users pay for the amount they use. In this case users are charged when they terminate the instance. On the other hand, providers use dynamic pricing where price is set based on a number of factors such as availability,

time, etc. For example, a user requesting an instance at peak time will pay less due to the system being fully utilized. If the system has less users then the price is going to be high[12]. One example of dynamic pricing is spot instance from Amazon. More on this later in this section when we describe into more details what spot pricing is.

The leaders of cloud computing are in a continuous competition to overcome one another. They are implementing new ideas for pricing in order to make it more attractive for users. For example, Microsoft Azure has recently introduced the pricing per minute to match the prices per hour provided by Amazon. In the following sections, we will describe the pricing models from Amazon and Microsoft Azure that are currently available.

2.3 Amazon Cloud

The leading cloud computing provider today is Amazon with AWS(Amazon Web Service)[5]. Amazon Web Service was developed in 2002 which only had a "suite of cloud based services"[15] such as storage, computation, etc. In 2006, Amazon EC2(Elastic Compute Cloud) became available as a commercial web service that allowed consumers and companies to rent computers. According to Jeremy Allaire, CEO of Brightcove, Amazon was the first cloud provider to offer IaaS[15].

Amazon provides various types of instances which consist of nine categories as we can see below[16]:

- Standard Instances - are available in small, medium, large and extra large options. The instances are used by consumers, with lots of resources at a low cost based on the type of applications required to execute.
- Second Generation Standard Instances - are available in extra large and double extra large options. This instance is used where the application requires a high demand of CPU and memory performance. They are used to manage "high traffic content management systems".
- Micro Instance - is a low cost option that provides a lower amount of CPU. The user will be able to increase the CPU capacity at short intervals when more cycles are available. This is used for low throughput applications that still requires significant compute cycles periodically.
- High Memory Instances - are available in extra large, double extra large and quadruple extra large instances. These instances give users between 17GB up to 68GB memory. They provide the lowest price per GB of RAM. Used mostly for database applications, distributed memory cache, etc.

- High CPU Instances - come in medium and extra large options. The instances are characterized by compute power. They have the lowest price per CPU ratio with respect to memory as well as providing the lowest cost per CPU. This instances are used for high traffic web applications, batch processing, video encoding, etc.
- Cluster Compute Instances - available in quadruple extra large and eight extra large. The instances in this category provide high CPU and network performance. The logical cluster provides high-bandwidth and low latency network between the instances in the cluster. It is used mostly for high-performance analytic systems. Commonly used in parallel programming.
- Cluster GPU Instances - provides only one type of instance that allows users to use NVIDIA Tesla GPUs using CUDA and OpenCL. They also provide high CPU and cluster network.
- High-I/O Instances - are used for high performance databases. The instances offer high CPU, memory and network performance.
- High-Storage Instances - used for very large data sets. They provide high storage density, low storage cost and high sequential I/O performance[17].
- EBS-Optimized Instances - in addition to high storage instances, users might choose to launch EBS-Optimized Instances using IOPS volumes at a lower cost.

The instances described above range from 0.060$/hour for the small standard instance to 4.6$/hour for the high storage instance. As of February the prices have been decreased for M1(First Generation Standard Instance), M2(High Memory), M3(Second Generation Standard Instances) and C1(High CPU) families of instances by an average of 10-20 percent as mentioned in [18]. Figure 3 shows how each region is affected by the price reduction. Since 2006 when it was introduced to public, Amazon has reduced their prices twenty-five times. However, with such a large variety of instances not all parts of the world can have access to all types of instances. This is not necessarily a problem, because users can choose the hosting site that they want regardless of the area they are in.

Amazon provides three different types of pricing based on the user requirements:

- On-Demand Instances
- Spot Pricing Instances
- Reserved Instances

Many providers these days tend to do offers for long-term commitment because it will be much cheaper for the consumer than on-demand pricing, however, the users

Region	Savings (%)				
	M1	M2	M3	C1 Medium	C1 Extra Large
US East (Northern Virginia)	7.7%	8.9%	13.8%	12.1%	12.1%
US West (Northern California)	27.7%	9.1%	-	11.3%	11.3%
US West (Oregon)	7.7%	8.9%	-	12.1%	12.1%
AWS GovCloud (US)	22.3%	9.3%	-	9.8%	9.8%
Europe (Ireland)	23.5%	9.1%	-	11.3%	11.3%
Asia Pacific (Singapore)	5.9%	2.2%	-	1.6%	1.9%
Asia Pacific (Tokyo)	4.3%	2.5%	-	2.6%	2.6%
Asia Pacific (Sydney)	5.9%	2.2%	-	1.6%	1.9%
South America (São Paulo)	30.4%	20.6%	-	13.0%	13.0%

Figure 3: How much money can a user save depending on the area he is using the resources.

must pay an up-front cost. However, the providers need to ensure that the computations of one customer does not affect the performance of another customer.

In the following part, we are going to describe different types of pricing which allows basic negotiation between the consumer and the provider.

2.3.1 On-Demand Instances

On-demand allows users to pay for as much as they use without any long-term commitment. This allows the user to rent and maintain hardware at a low cost. However, Amazon charges by the hour, even though a consumer used only one and a half hours, he/she will still be charged for the two full hours. This type of pricing is mostly used for applications that require short computations. In case users need to run the application for a very long period of time such as a year, he/she will be better off by choosing a long-term commitment. This has been proven later on in this section to be cheaper than using on-demand pricing for a full year. The pricing for each instance in this case is set according to the complexity of the instance. The instances that do not involve GPUs or lots of CPUs will be automatically cheaper since they are mostly used for simple development and not for high performance.

2.3.2 Spot Pricing Instances

Spot Pricing allows users to reserve for a period of time the unused space in the cloud at a much lower rate than the on-demand pricing. When using spot pricing instances, the user has to bid the biggest price he/she is willing to pay in order to run the

instance(s). Amazon provides a history of how the market price changes over time. In Figure 4 we can see how their pricing fluctuated over a 3-months period with respect to On-Demand which is constant.

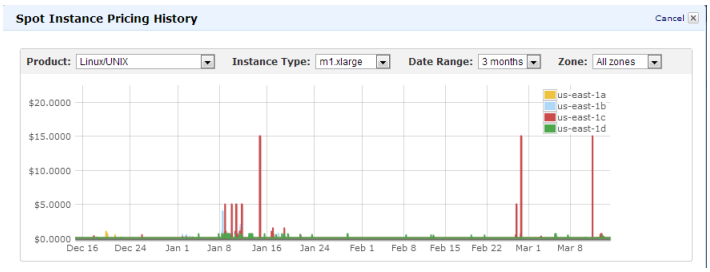

Figure 4: This is a graph that states how the Amazon price market for spot instances has changed over the last 3 months.

The user makes an estimated guess about how long the instance will run and how much it will cost to run it. However, if the market price goes over the users' bid then the instance will be terminated instantly. In this case, if the instance has only ran partially, the user will not be charged for the extra hour because it was terminated by Amazon. However, the user can terminate the instance when the computation is done, in which case the user will be liable for paying the full hour if it only ran partially.

Using this type of pricing the user can save money when doing computations. However, not everyone will be willing to bid in order to run the instances because they might have very important computations in which case they do not afford to lose time and money when the instance gets cut off. The users should bid strictly what they want to pay for the instances. For example, in the past people have bid higher amounts until the price went up to 100$. Table 1 presents the lowest price market the Spot Price can have for a specific instance:

Users that require to run an instance for only a small amount of time, might find it cheaper to use spot pricing because they can save up money. In case the spot price goes over the on-demand price they can choose to opt out of the spot instance and use on-demand pricing without terminating the instance.

2.3.3 Reserved Instances

The last pricing model offered by Amazon is the long-term commitment. This is for users to ensure their resources at any time they want. Using a contract gives a user the benefit that those resources will be there at all times as well as saving a great deal of money. So by doing a long-term commitment users get a lower rate for the

Instance type	Lowest Spot Price	On-Demand Price
Standard Instances		
Small	0.007$	0.06$
Medium	0.013$	0.12$
Large	0.026$	0.24$
Extra Large	0.052	0.48$
Second Generation Standard Instances		
Extra Large	0.0575$	0.5$
Double Extra Large	0.115$	1$
Micro Instances		
Micro	0.005$	0.02$
High-memory Instances		
Extra Large	0.035$	0.41$
Double Extra Large	0.07$	0.82$
Quadruple Extra Large	0.14$	1.640$
High-CPU Instances		
Medium	0.018$	0.145$
Extra Large	0.07$	0.580$
Cluster Compute Instances		
Quadruple Extra Large	3.0 $	1.300$
Eight Extra Large	0.27$	2.400$
High-Memory Cluster Instances		
Eight Extra Large	0.343$	3.500$
Cluster GPU Instances		
Quadruple Extra Large	0.4$	2.100$

Table 1: The table presents the differences between the lowest price the Amazon Spot Price can have and the On-Demand price that the users can opt to choose. The data is based on the utilization they had on the 25th of June.

instances they want to use. However, the drawback is that they are required to pay an up-front cost which is for reserving the instances. But in the end, the price is lower by almost 50% as we can see in section 3.5. Amazon provides contracts for one-year or three-year commitment. Based on this the prices for the instances can vary. In case the user goes for one year with low utilization the price per instance will be the highest, while if he/she goes for three year contract with high utilization the instance price is the lowest. However, if users do not need high utilization then they only pay for what they use, otherwise the user pays even if the instances are not in use at all times. Since the prices for those are lower the user is not affected. The upfront cost depends as well on reserved instance type(low, medium or high utilization). Additionally, users may get

more discounts if they go over a certain budget as stated in the table below:

Total Instances Reserved	Upfront discount	Hourly discount
<250000$	0%	0%
250000$ to 2000000$	10%	10%
2000000$ to 5000000$	20%	20%

Table 2: The table shows the discount that is applied for the reserved instances based on budget.

2.4 Microsoft Azure

Microsoft Azure holds the second leading position after Amazon. Azure has only been introduced since January, 2010 but only worked as a fully paid service since February, 2010[19]. Azure supports three types of services: Virtual Machines(IaaS), Cloud Services(PaaS) and Web Sites(PaaS)[20]. One of the most important advantage that Azure provides is the ability to support multiple models. For example, the user can have a web site to present data and a VM to generate the data. Azure used to provide various types of instances at an hourly price like all the other providers. However, they just recently introduced the pricing per minute which they state is better from the users point of view since he/she is charged for the exact amount being used. In the next section, we will analyse whether it is better for the user to pay by minute or by hour. Another thing from which Azure users can benefit is the 1-month free trial where users get 200$ to use however they want. It offers users a way to look around and see how the cloud behaves without having any strings attached. Even though, Amazon provides a free trial as well, for a year long, it does not give the users the liberty to choose what they want. Amazon's free-trial is more constrained allowing only certain instances to be used in which case the user will not know the performance of other instances.

Microsoft Azure provides only two categories of instances as follows:

- Standard Instances - these instances provide only an "optimal set of compute, memory and IO resources". These contain 5 different types of instances based on the users needs.[21]
- Memory Intensive Instances - which provide high amounts of memory, they are usually used for databases. The instances have 28GB and 56GB RAM memory.

Microsoft Azure has only two types of pricing:

- pay-as-you-go
- monthly plans

2.4.1 Pay-as-you-go price

The pay-as-you-go pricing works in a similar way with the one provided by Amazon. Here, the users can request as many instances as they want without having any upfront cost or long-term commitment. Currently, Microsoft offers pricing per minute which will be compared in Section 2.5.1 with the pricing per hour provided by Amazon. However, the user is restricted to only 744 hours per month as mentioned on Microsoft Azure Pricing Website[21]. For the pay-as-you-go plan users can terminate the instance at any time being charged just for the period of time used. The pricing per minute matches the Amazon price per hour.

2.4.2 Monthly Plans

Microsoft Azure also provides monthly plans for users that require more time to run their applications. They offer two plans: a six-month commitment and a one-year commitment. These plans in turn can be of two types: pay monthly or pre-paid. With a pay monthly plan the user must commit with a minimum of 500$/month, while with pre-pay, the user has to do it for the whole period of time. In this cases, the price per instance varies based on the commitment level spend, the duration of the commitment and whether they are choosing a pay-monthly or pre-paid plan. Any unused resources that a user has at the end of the monthly bill will be rolled over as credit for the next month and so on until the end of the plan. In case the consumer goes over the monthly commitment he/she will have to pay for any additional resources at the pay-as-you-go price. If the user decides to go for a monthly commitment, then he will have the current month commitment plus any left overs from the previous months. Otherwise, in case of a pre-paid plan the users' monthly commitment is equal to the remaining balance made on subscription. For users that have opted for monthly plans the billing is going to be done at the beginning of each billing cycle. Users can change their monthly commitment at any time without changing the plan length. In case the new commitment falls under a higher rate of discount, then the discount will be applied in the next billing cycle.

Microsoft Azure offers discount rates for their monthly and pre-paid plan as seen in Table 3.

2.5 Comparing Amazon vs Microsoft Azure

This section compares the cloud computing providers mentioned above. As we can conclude from the sections above, both Amazon and Microsoft are well known cloud computing leaders. Amazon is one of the oldest cloud computing provider which offers

Monthly Commitment Spend	6-Months Plan(Monthly Pay)	One-Year Plan(Monthly Pay)	6-Months Plan(Monthly Pay)	One-Year Plan(Pre-Paid)
500$ - 14999$	20%	22.5%	22.5%	25%
15000$ - 39999$	23%	25.5%	25.5%	28%
<40000$	27%	29.5%	29.5%	32%

Table 3: The table shows the discount that is applied for the monthly plans provided by Microsoft.

users more stability and reliability. Microsoft has become known a couple of years after, but still managed to get the second place in the leading market. Amazon and Google stated that the cloud industry has started to behave more like a utility market, making it harder for the providers which now have to race on prices[22].

As of recently, Microsoft announced that they will match the pay-as-you-go standard instances with Amazon Standard Instances from June, 2013. Therefore, the prices will suffer a reduction between 21 and 33%. Mike Neil, a general manager from Microsoft Azure has told "The Register"[22] that they are trying to make it in such a way that the pricing will no longer be a deciding factor for consumers. However, by introducing pricing per minute Microsoft has brought some benefits for the users regardless of the fact that the price per hour is the same. There is still a difference in the price for the memory intensive instances as seen below.

Instance Type	Amazon	Microsoft
A6(Memory Intensive)	0.82$	1.02$
A7(Memory Intensive)	1.640$	2.04$

Table 4: The table shows differences in pricing for the memory intensive instances on 27th of June. The Microsoft prices have been updated on 16th of April, 2013, while Amazon has left the same prices since February, 2013.

However, even though the pricing per pay-as-you-go matches, Amazon provides a cheaper price for reserved instances as we can see bellow in Table 5. When reporters asked whether they would try to match those as well the answer was no. They stated that their commitment is on the pay-as-you-go model and if users want to get further discounts they can sign up for a Microsoft Enterprise Agreement.

Another difference between the two providers is that Microsoft provides six-months or one-year commitment while Amazon provides one-year or three-year commitment. In both cases users have to pay an upfront commitment just the length of the

Instance Type	Amazon	Microsoft
Extra Small	N/A	0.01-0.02
Small	0.021$	0.04-0.05
Medium	0.042$	0.08-0.1
Large	0.084$	0.16-0.19
Extra Large	0.168$	0.33-0.38
A6(Memory Intensive)	0.206$	0.56-0.66
A7(Memory Intensive)	0.412$	1.12$ - 1.31$

Table 5: The table shows differences in pricing for medium standard reserved instances from Amazon with the Monthly commitment from Microsoft.

contract is different.

Microsoft has recently announced that they have 200000 consumers using either trial or pay-as-you-go model while Amazon has millions of users each year. Another thing that differentiates Amazon from other companies is that it comes from a retail background applying a retail economics perspective to the cloud including price-cuts, only developing technologies when they have a demand, etc.[23] Amazon is big which makes it easier to monitor what jobs are running and launching new ones based on the users demands. They now have the ability to see what is being used by application developers. Competitors find it hard to beat someone like Amazon. Because Amazon is big it can attract more consumers. More and more companies are using Amazon cloud as a platform to develop new applications. For example, Red-Hat has decided to build its platform-as-a-service on top of Amazon's infrastructure[23]. Amazon is targeting everyone from simple developers to world's biggest enterprises. Since it first came out Amazon revenue has increased significantly as we can see in Figure 5.

Azure subscriptions have gone up by 48% in the last six months[24]. However, according to Bloomberg, Microsoft has earned 1$ billion in revenue in the last year. This is the most Microsoft has ever made. In a recent study, it was discovered that 20% of the companies use Azure, while 71% use Amazon. However, it is estimated that Microsoft might rise to 35% in the next year. In the last couple of months Microsoft and Amazon have been battled on the price.

Based on the information that was provided above we can make a comparison between Amazon and Microsoft pricing. First of all, we will consider a normal user that just requires to use one medium instance for an hour and a half. If the user chooses Amazon he/she will pay 0.24$. When using Amazon, a user is going to pay for a full hour no matter how long the instance was used for. If the user chooses Microsoft, then the price will be 0.18$ since they charge per minute. The other case to be considered is when there is a company that wants to run 1000 instances for an hour and a half. When

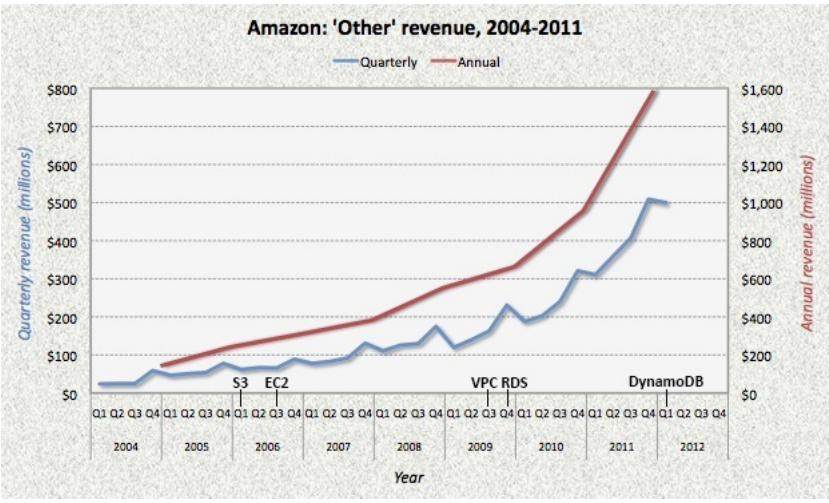

Figure 5: The graph shows the dramatical increase since Amazon has introduced EC2 and S3. The increase can be seen year-by-year with an intense growth curve based on the expansion and addition of new technologies.

using Amazon, the company will pay 240$, while for Microsoft they will have to pay 180$. From here we can conclude that for a normal user it does not make a difference to which provider he/she chooses. The difference will be when the consumer requires to do large computations for a longer period of time. In this case the user will be better off to use Microsoft since he/she can save up money.

As seen above, one of the most interesting aspects that we are going to take a look is the way they charge users for the resources requested. The two leading providers offer the same price per hour. However, we have mentioned above that Microsoft Azure charges users per minute as of July, 2013 while Amazon charges per hour. This is one of the most interesting features offered lately that has raised many questions. Does it actually benefit the users in any way? In the next part of this section, we will make a comparison between the two pricing models to see whether the user can save up money by using one or the other. To make the comparison accurate, we are going to take a look at three cases: worst case, best case and random case, as well as from the view of a normal user and a company that requires more resources. As mentioned in the beginning, Microsoft has stated that if the units are low the user can get more benefits out of it. This is why we are going one step forward and try to analyse whether introducing pricing per second can be more beneficial than the pricing per minute.

2.5.1 Price per minute vs Price per hour

Every provider today offers various plans for the users to choose from. One of the plans is to pay on-demand for the exact amount that they use. When Amazon first started they introduced the pricing per hour which they are still using today. Recently, Microsoft Azure has introduced the price per minute stating that the users will get more benefits. This will make the pricing more accurate since the users will pay for the exact amount they use. According to Microsoft, as lower the measuring unit is, as many benefits the consumer has. Therefore, this section will consist of analysing the pricing per minute introduced by Microsoft with the pricing provided by Amazon. The comparison will examine the best, worst and random case, in order to make an accurate assumption regarding which one is more beneficial for the user.

First, we will make a couple of assumptions since we can only relate to one instance which is provided by both Amazon and Microsoft Azure. As we can see in Figure 6, the instance that we are going to use to compare the price is based on the medium instance.

	AWS	Google Compute Engine	Windows Azure
VM Configuration	1 virtual core, 3.75 GB memory	1 virtual core, 3.75 GB memory	2 virtual cores, 3.5 GB memory
OS	Linux	Linux	Linux
Price per hour	$0.12 (US East), $0.13 (US West)	$0.132	$0.12
Billing granularity	By hour	By minute, with a minimum of 10 minutes	By minute, No minimum, No billing for less than 5 minutes

Figure 6: This Figure shows the advantages of pricing per minute compared to one from Amazon which is per hour. The pricing has been done for the same instance from Amazon, Microsoft and Google[34].

As mentioned in [35], the prices from Microsoft will match the ones from Amazon for the "base-level infrastructure - storage, compute instances, etc.". According to Microsoft the change can benefit the user on the long run. Therefore, we devised some analysis to see how much does it really impact the consumer and whether it is worth it. Taking a look at the model, the consumer is only impacted if the instance is shut down before the hour elapses.

The analysis is conducted based on the information in the picture above. We know that the price per hour for both Microsoft and Amazon is 0.12$/hour. From here we can deduce that the Microsoft price is 0.002$/min. During the analysis the best case,the

worst case and the random case are evaluated based on different perspectives. The best case emphasizes Microsoft as being much cheaper than Amazon(the consumer uses only a couple of minutes from an hour), while the worst case leaves things to the latitude of the user to choose the provider they think is more stable and reliable since the price difference is going to be very low. The best case from the user's point of view is when the instance gets stopped after 61 minutes. The worst case is when the consumer uses the resources for 1 hour and 59 minutes(119 minutes per total). For the random case, a random generator set has been used because the users will most likely end the instances randomly. This is one of the most accurate comparisons. The difference between prices is only vizible in the [1..59] interval since per hour the price stays the same.

Let's first see how many benefits a single user can have when using 10 resources. The comparison is done taking into consideration all three possible cases. First, we consider the best case where the user stops the instance after 61 minutes. In this case the user only pays 1.22$ when using Microsoft Azure rather than 2.24$ required to pay for an Amazon instance that charges by hour. The user saves 1.18$ which he/she can use to get more instances later on (up to 9 extra hours to spend on the cloud). The next case to be considered is the worst case, where the users do not get many benefits out of it. Considering that a user ends the instance after 119 minutes, he/she needs to pay 0.002$ less when using the pricing per minutes rather than what they would normally pay for a normal hour. Therefore, the users can decide to choose the provider that is more reliable and stable and well known on the market. The last case emphasizes a random case which is more likely to occur because the users are going to stop the instances at random times. Using a random generator set to simulate 10 randomly numbers between [61..119]. The random set provides an average time of around 90 minutes. The user gains in this case 0.6$ when using Microsoft over Amazon. This is not a big price difference, however, the user can still get 5 extra hours to spend another time with any of the providers.

To conclude the above analysis, we can see that the users can get between [0,002..1,22]$ depending on how long they have used the resources for. Looking at Amazon the consumers need to make sure they are aware when the hour "flies out" so they can close the instance in case they finished their job. Sometimes it is hard to ensure when the full hour has elapsed and they might go over 1 hour by a couple of minutes which will still charge them for the following hour. Depending on how long the consumers need to use the resources, they should made a decision based on what exactly they need and what they think it is more important. If they want reliability then Amazon is the cloud that has been on the market for a longer period of time, making it more stable. In this case, it is more about the requirements rather than the price because the difference is very small even if we look at the random case.

We have seen above how the prices vary for a normal user, now we will take a look at the impact it has on a large company that requires 100000 resources at a time.

Following the same principle as above we will start by looking at the best case. Using this case, the company can save up to 11800$ if they choose Microsoft, as we can see in Table 6. Clearly, this can save them lots of money, that they could be using for more complex instances. However, this case is less likely to happen so we will take a look now at the random case. For the random case, we have seen above that the average for the random set is 90 minutes. Taking this into account, we can compute the money that a company can save which is up to 6000$. Again, for a big company this is a lot of money that can be used for something else that maybe requires more important things to be done. Last but not least, we considered the worst case, in which the company can only save 200$. In this case, the company can choose either provider and probably the most stable one.

From the above analysis, we can conclude that the company can save up between [200..11800]$. This can be beneficial for the company because the money can be used for something else or to buy more hours for computations. The random case still gives the company a gain of 6000$ which can be used further for getting more instances, so the company benefits from the pricing per minute.

In the following table we will show a brief overview of what it was described above, in order to have a clear picture of everything that was illustrated above.

Case	Provider	Price per minute	Number Resources	Price Resources
Best Case(61 min)	Microsoft Azure	61*0.002$/min=0.122$	10	1.22$
Best Case(61 min)	Amazon	2*0.12$/hour=0.24$	10	2.4$
Random Case(90min)	Microsoft Azure	90*0.002$/min=0.18$	10	1.8$
Random Case(90 min)	Amazon	2*0.12$/hour=0.24$	10	2.4$
Worst Case(119 min)	Microsoft Azure	119*0.002$/min=0.238$	10	2.38$
Worst Case(119 min)	Amazon	2*0.12$/hour=0.24$	10	2.4$
Best Case(61 min)	Microsoft Azure	61*0.002$/min=0.122$	100000	12200$
Best Case(61 min)	Amazon	2*0.12$/hour=0.24$	100000	24000$
Random Case(90min)	Microsoft Azure	90*0.002$/min=0.18$	100000	18000$
Random Case(90 min)	Amazon	2*0.12$/hour=0.24$	100000	24000$
Worst Case(119 min)	Microsoft Azure	119*0.002$/min=0.238$	100000	23800$
Worst Case(119 min)	Amazon	2*0.12$/hour=0.24$	100000	24000$

Table 6: The table concludes what it was previously described and how the prices per minute vs per hour vary depending on the cloud provider and the case chosen.

The price per minute can have advantages for both a simple user and a company. Only in the case where the consumer just wants to use a simple instance for an hour, it is not worth all the hassle of deciding and they can just go with the cloud that they think

it is more reliable. After doing all these research, one of the questions this raised is why the providers have not come up to an even more exact pricing like per second. This will be explained in the next Section 2.5.2.

2.5.2 Price per second vs price per hour

Section 2.5.1 described how the pricing per minute affects the user. The conclusion deducted after analysing the results was that the price per minute can really save the user money. It all depends whether the consumer was happy with the services provided. In this section, we describe the effects that pricing per second has on a simple user and a company. Using the same principle as above and knowing that a price per hour is 0.12$ brings us to the conclusion that the price per second will be 0.000033$.

Let's consider now how the pricing per second works for a normal user. For the best case, we can see that the user saves 1.19967$ when using the pricing per second. In this situation, the user can save up money to buy resources later on. The next case that we are going to take a look is the random case which is going to be around 4993 seconds when using a random generator set. Now the user can save up to around 0.8$. The last but not the least, we considered the worst case in which the user will only pay 0.00033$ less. This is not worth it, in which case the user should go with the provider that is more reliable and stable.

From analysing the above the user can save between [0.8..1.19967]$. From here we can conclude that there is no difference between using seconds or minuted for a normal user. For this reason, there is no need to implement a model for developing a pricing per second for a normal user.

We have seen that the pricing per second does not have any impact for the normal user. To make the analysis more accurate we took a look whether a company can benefit from this pricing scheme or not. Considering again the three cases, we started analysing the best case. The company can save up to 11997.7$ by using the best case. In this case, the difference between pricing per minute or per second is only 100$. Considering now the random case, the company can redeem around 7500$ while for the worst case the company gets 3.3$.

From the analysis above the company can save up between [3.3...11997]$ based on when the instances are terminated. Table 7 shows an overview of what it was described above.

In comparison with the per minute pricing, it is not better. In this state there is no point in implementing the latter one from which not even the big companies can benefit. They can save up money but it is not worth the hassle of implementing a new pricing scheme.

Case	Provider	Price per sec	Number Resources	Price Resources
Best Case(3601s)	Microsoft Azure	3601*0.000033$/s=0.120033$	10	1.20033$
Best Case(3601s)	Amazon	2*0.12$/hour=0.24$	10	2.4$
Random Case(4993s)	Microsoft Azure	4993*0.000033$/s=0.164769$	10	1.64769$
Random Case(4993s)	Amazon	2*0.12$/hour=0.24$	10	2.4$
Worst Case(7199s)	Microsoft Azure	7199*0.000033$/s=0.239966$	10	2.39966$
Worst Case(7199s)	Amazon	2*0.12$/hour=0.24$	10	2.4$
Best Case(3601s)	Microsoft Azure	3601*0.000033$/s=0.120033$	100000	12003.3$
Best Case(3601s)	Amazon	2*0.12$/hour=0.24$	100000	24000$
Random Case(4993s)	Microsoft Azure	4993*0.000033$/s=0.164769$	100000	16476.9$
Random Case(4993s)	Amazon	2*0.12$/hour=0.24$	100000	24000$
Worst Case(7199s)	Microsoft Azure	7199*0.000033$/s=0.239967$	100000	23996.7$
Worst Case(7199s)	Amazon	2*0.12$/hour=0.24$	100000	24000$

Table 7: In the table above we have shown how the price varies based on different providers and different pricing scheme. From this table the users can conclude which provider to choose in terms of lowering the price.

3 Negotiation

We have mentioned above in Section 2 that the pricing models can be improved. One way to do so is to create a new mechanism that benefits both providers and consumers. We are going to start by giving an overview about some negotiation mechanisms and strategies, followed by the description of a new mechanism based on interactive negotiation. Negotiation is founded on the following attributes: price, availability of resources and utilization. It has the advantage of providing benefits for both sides in case the negotiation has been settled as well as increasing the number of users. After describing the mechanism, we developed a design and an implementation that will help us analyse whether the interactive negotiation is better or not. The last part of the section describes a new pricing model that is going to emphasize a pre-paid plan in which users do not require to have a contract.

3.1 Introduction to Negotiation

These days, negotiation is one of the processes that a lot of companies acquire in order to thrive. This consists of an agreement between the consumer and the provider that will equally benefit both. More and more companies are using negotiation because it is the key point for business survival, growth and empowerment. Negotiation is seen as a time spent to make profit and cost effective relationships with other companies that will potentially promote the negotiators reputation. When the customers are happy, it will automatically attract more customers, through word of mouth or other means, making the company grow and remain competitive in the market. The way to remain in the competitive market is by embracing negotiation which differentiates businesses between each other[25].

Every business is aware that in order to survive it needs to produce money. Having a consultant that can deploy high negotiation skills brings benefits to the company. It helps keep the business up by making wise decisions in terms of vendors and what deals fit best the business[26]. Negotiation is often seen as a finite state machine.

In case a company decides to go for a negotiation model, they need to know what are the key elements for a negotiation process[31]:

- Making a deal through a final offer from buyer or seller;
- It needs to make sure there are participants such as customer and provider;
- There are messages send between the participants to come up with a deal suitable for both sides;
- The process flow describes the changes in the deal;
- Send back messages to the participants when the deal changes.

While a lot of companies are using negotiation, cloud computing negotiation is still in its early days. The market leaders use a very simple negotiation strategy, in which the customer is presented some prices from which he can either chose to accept or reject them. Cloud Computing is becoming one of the modern technologies these days, as mentioned in Section 2 which drives the prices downwards by any means in order to stay in the competitive market[33]. There are more providers which are entering the "playing field each trying to differentiate itself from the already established players"[33]. One way to look at this is to choose different pricing schemes in order to keep the business growing.

3.2 Negotiation Mechanisms

Firstly, we took a look at some of the negotiation mechanisms that are used these days in various subjects such as grid computing, business, etc. A negotiation protocol[2] is initiated in all the negotiation strategies, in which one presents an offer and the other party either accepts it or sends back a new proposal. This is known as an alternating offer as mentioned in [27]. However, time constraints should be specified since this can go on for months. Various negotiation strategies[3] have been enforced[28]:

- Contract Net
- Auction Model
- Game Theory Based
- Discrete Optimal Control Model

The negotiation strategies are selected based on the type of business that each provider is running. The provider needs to make an elaborate analysis of the negotiation mechanisms available to ensure that he/she has selected the proper one for the business that can bring benefits on the long run.

The first negotiation strategy discussed in this dissertation is the *Contract Net* protocol which is used to exchange the Service-Level Agreement(SLA) between the provider and the customer[27]. In Contract Net, an agent acts as a manager evaluating the task announcements from other agents and bids for the tasks that the agent is engaged in. This type of protocol can have only two possible bid outcomes: *accept* or *reject*. Therefore, Contract Net is convenient for multilateral processes. However, it has some drawbacks as well. The protocol cannot provide feedback for an acceptable agreement between the two parties. Because of this the protocol is not useful for bilateral process. Therefore, it will not be of any use in the rest of the dissertation, since the main reason of the paper is to come up with a new negotiation mechanism that can benefit both parties through communication and interaction.

The second negotiation mechanism is *Auction*, one of the most used mechanisms these days. Auctions are used mostly for antic pieces, painting, grid computing, etc. This model of negotiation allows both parties to communicate, they are either in the same room making bids or separately in case the bid is sealed. Auctions are generally, one-to-many negotiation, meaning that a seller can have many buyers, or vice versa as mentioned in [28]. For negotiation, both parties need to agree on the offer without any penalties in case of rejection. There are different types of negotiation:

[2]In this dissertation, negotiation protocol will be considered as a set of norms that constrain the proposal of negotiation.

[3]The mechanism through which it maximises its own individual welfare.

- English Auction - it is an ascending auction. The bidding starts at the reservation price and progresses until only one bidder is left. This is equivalent to the Vickrey model. The resources are given to the user that values them the most. The winner claims the item at the last bidding price.
- Dutch Auction - it is a descending auction. The auctioneer begins with a price too high for any buyer and then progressively goes lower on the price until someone requests it. The winner takes the item at the price it was bid. This is not necessarily effective because users might still end up paying more than what the item costs.
- First-price, sealed bid Auction - The bidders submit the price they are willing to pay. All the prices are opened at the same time and the winner is the one who is willing to pay the most.
- Vickrey Auction - Bidders submit a single irrevocable, sealed bid. They are not aware of what the others bid. The bids are all opened at once and the winner is the one that is willing to bid the most for the item, and he/she will pay the second-highest price that was bid.
- Multiround sealed bid Auction - each round has a deadline at the end of the round. It can either start a new one or the auction is closed. This might take too much time.
- Electronic bulletin board approach - bidders can check the bulletin board and they have a chance to bid over the highest bid. This is common for Internet based where people are all over the globe.
- Yankee Auction - the participants pay the amount they bid.
- Non discriminative Auction - refers to the bidders that won the auction pay the lowest bid rather than what they bid for.

When creating an auction there needs to be some policy choices applied[31]:

- Anonymity - in a sealed auction the number of items being auctioned and the number of items bid for each bid, don't need to be revealed. Only the winning bidders could be revealed. The inventory should not be disclosed. However, in an open bid all these could be exposed beforehand.
- Restrictions on bid amount - Seller can specify the minimum starting bid. If the proposal happens regularly for an item then the minimum bid can be calculated based on the history known.
- Rules for closing the bid - it can have a closing time or it can go on until some certain time is elapsed between the bids. It can also be closed when the items have

been sold.

- Evaluation rules and breaking tied bids - when an item is sold a higher bid is better than a lower bid. If the two bids have the same quantity then it can be given to the one that arrived first.
- Services provided to sellers and bidders - reserved prices is one thing that can be provided to the sellers by the auctioneers. Another one can be to "certify the quality of the product, collecting payment on behalf of the seller", etc.

To conclude what it was described above, the auction model is used mostly for dynamic resource allocation. Therefore, the negotiation strategy going to be used in this dissertation embraces the Vickrey model. The reason for choosing this auction model, except the fact of being dynamic, is that it can be inference-proof. By inference-proof negotiation we can understand that if an auction is set properly neither the buyer nor the seller have to lie about their strategies in order to win. The other auction model that will be used is the Yankee auction. This is used when there is only one user bidding at a time.

Another negotiation strategy is the *Game Theory Based Model*. In this model, each agent has the responsibility to offer a plan, if the plan is accepted by all users then the negotiation is terminated. If agents choose "out" or no agreement is made then the process is turned down.

The last negotiation strategy discussed in this paper is *Discrete Optimal Control Model*. This is optimized for the market model. The *Discrete Optimal Control Model* consists of three steps as specified in [28]: getting all the information together, coming up with a decision and making the decision known. The information acquired can be about price, tasks, the number of tasks, etc. After gathering all this information the marketplace agent can decide what algorithm will be appropriate. In the end, the decision can be made available to other related agents. The mechanism has m processors and n tasks. Some assumptions and constraints need to be made according to the needs of the users. The mechanism is meant to give the optimal solution.

3.3 Negotiation Strategies

We have talked about negotiation mechanisms that are available, now we are going to take a look at the negotiation strategies that we can adopt. Current negotiation mechanisms support one-to-one negotiation as described in Section 2, however, many-to-many transactions can be done as well[32]. The latter is possible by using one-to-one negotiations and then everyone reports to just one agent which will then make a decision based on their next move. Negotiation can have many attributes to negotiate over, but

for the purpose of this dissertation only price will be taken into account. Negotiation is usually done in the presence of incomplete information, since for our case the consumer will not know how many resources the provider has available. The utility function decides whether the negotiation between the user and consumer can be done. If minimum utility is ensured then a negotiation can be placed(if both areas of acceptability overlap then a solution can be found):

$$v(x_1, ..., x_n) = \sum_{i=1,...,n} w_i v_i(x_i), \sum_{i=1,..,n} w_i = 1 \tag{2}$$

where:

- x_i - is the i^{th} attribute of negotiation
- v_i - is the utility function
- w_i - is the weight or priority of the attribute i^{th}.

The one-to-many negotiation resembles multiple consumers that want to buy from a provider. In order to accomplish this, they require to exchange information, which involves interaction between the two parties. The one-to-many negotiation is done by making multiple one-to-one negotiations. However, in order to do that, the consumers are split into multiple agents which will be called sub-negotiators like in Figure 7. In this dissertation, we are going to develop a negotiation strategy using the one-to-many strategy.

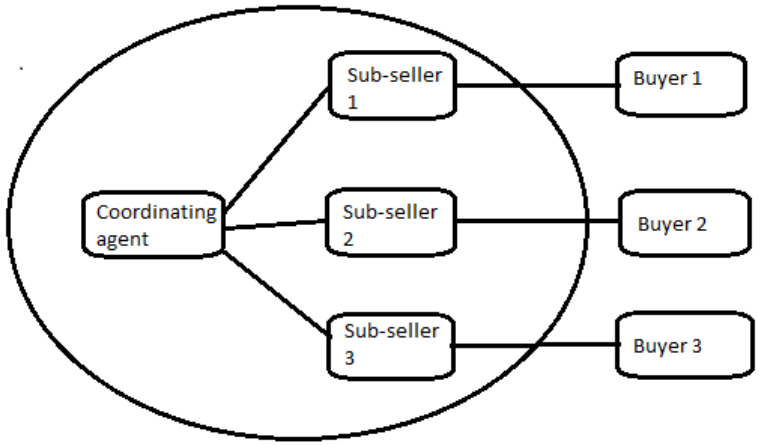

Figure 7: The pictures shows how the process of one-to-many negotiation works.

A few strategies have been developed that handle one-to-many negotiation:

- Desperate Strategy - in this strategy, time is important, which makes the deal to be sealed fast. If one offer is found then the sub-negotiators are terminated and the offer is accepted. If more than one offer is found, then the one with the highest utility is chosen and the rest of the sub-negotiators are terminated.
- Patient Strategy - this strategy waits until all the sub-negotiators have an offer and then chooses the best one available. However, there is a draw back because the strategy does not have a set time which means that it can go on for too long.
- Optimized Patient Strategy - in which an agent uses the information from one negotiation outcome to influence the others. It also ensures that no offer that is worst than an offer received is offered.
- Strategy Manipulation Strategy - which allows the coordinating agent to modify the negotiation strategy at runtime.

Looking at the strategies above, we will use the desperate strategy along the dissertation because time is important and the users will be satisfied if they can get the instances requested as soon as possible. In a negotiation process, the provider wants to maximise the revenue while minimising the risk of penalties, meantime the consumer wants to gain the maximum guaranteed for meeting the quality of service requirements at the lowest possible cost. In order to ensure that the quality of service has been met, there needs to be a negotiation-aware scheduler and a negotiation client. The scheduler is aware of the negotiation client only as a medium for submitting proposals or receiving feedback. Requests for smaller number of nodes was tested to have a higher potential of acceptance.

Like any other strategy, every negotiation model should have the following properties[29]:

- Guaranteed Success - making sure that an agreement will be reached;
- Maximising Social Welfare - this will ensure that the outcome will maximise the sum of utility function mentioned above for every negotiation participant;
- Pareto Efficiency - meaning that is not possible to find another solution that one party can be better off and at the same time no party will be worse off[30];
- Individual Rationality - this is ensured if playing by the rules is in the best interest for the participants. Without them the users will not engage in negotiation;
- Stability - means that the users should all behave in a specific way. One example of stability as mentioned in [29] is Nash equilibrium, in which the agent *s* can not perform any better than the agent *s'* and vice-versa;
- Simplicity - means that the participant can determine easily the optimal strategy;

- Distribution - it needs to ensure that there is no point of failure and that it provides good communication between agents.

After doing the analysis of the new negotiation model, we will determine whether the model supports all the above properties.

Negotiation is possible for fixed prices as well as auction models. For fixed prices, the seller makes an offer and the buyer can then accept it or the seller can withdraw the offer from negotiation[31]. This is the negotiation plan used currently in cloud computing. For example, if we take a look at Amazon they offer consumers on-demand pricing which they can chose to take it or not. For the auction, there is a deal available if the highest bidding price is above the reservation price, when being set by the bidder(someone who makes an offer). Currently, cloud computing does not offer interaction for setting the price on the resources used. Therefore, this dissertation will describe later in this section a new mechanism that will be able to handle auctions between the provider and the consumer.

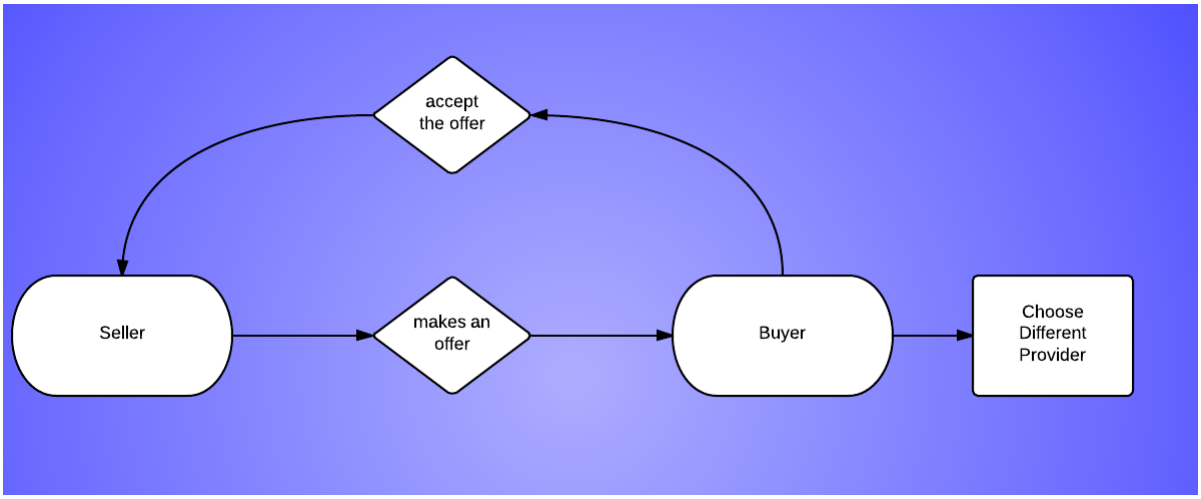

Figure 8: The Figure displays the basic negotiation model seen these days in cloud computing.

In the next part we describe the new negotiation model designed later on. The model is based on the Vickrey auction model defined above, as well as the Yankee auction. After the model is described, we are going to take a look at another approach for the pricing model. Cloud computing is similar with the pricing models provided by the mobile carriers. However, mobile carriers have introduced, a while ago, a new pre-paid pricing model. In this plan the users do not need to pay an front cost or to have a long-term commitment. Based on the information we have about the pre-paid plan for the mobile carriers, we analyse whether it is good for a cloud or not.

3.4 A new negotiation model

As we have mentioned in Section 3 above, negotiation is done in many fields with the purpose of making the company grow. However, cloud computing only handles simple negotiation which is mostly one sided as seen in Section 2. Therefore, in this section we will describe our negotiation mechanism that was designed and implemented as part of this work.

Firstly, a brief introduction about the basic negotiation models that are now available having Amazon as the main example. Amazon is the current leader in cloud computing, offering users on-demand, reserved and spot instances. The negotiation model in this case does not require any interaction. The consumer can decide to take it or leave it or maybe use a different provider. This simple model of negotiation has brought growth to the company and a very good reputation. Amazon is known for having the lowest prices for on-demand pricing strategy as well as being reliable. Therefore, it only attracted more consumers since it started being on the market. One of the things that a cloud should have is a small price from which the users can benefit and that will automatically increase the utilisation. Also by providing the users with different types of instances gives them more flexibility in choosing what they need without purchasing the hardware. Of course, every instance has a different price but it is still cheaper and time effective than getting your own hardware. Amazon also provides reserved instances for which users have to pay an up-front cost plus any instance they use depending on the utilization level. This is mostly used by consumers which are using lots of resources in order to decrease the price they pay.

Since none of the models available offer interaction with the user, we devise a model that uses negotiation through auction. In auction models, the users interact with the provider. The negotiation mechanism consists of a bidding mechanism for the user. The mechanism is using the Vickrey and Yankee approach described in the beginning of the section. If the negotiation is done between a single user and the provider, then the Yankee auction strategy is going to apply as long as the price bid is between the low price and the maximum price described below. This is illustrated in Figure 9

In the Vickrey mechanism, if there are multiple users bidding at the same time for the same instance then they are going to bid against each other. Otherwise, each one bids for the instance. The provider needs to make an estimate of how much energy they used for powering the resources as well as how much does it cost them to buy more resources. This should be done for the types of instances that are most utilized. After setting those prices, the provider can create their price that just contains the maintenance of the instance. This is the minimum price from which the provider cannot get any profit but in the same time they do not lose anything. The providers' perspective should be to minimise the price as low as possible in order to attract more users. They should then

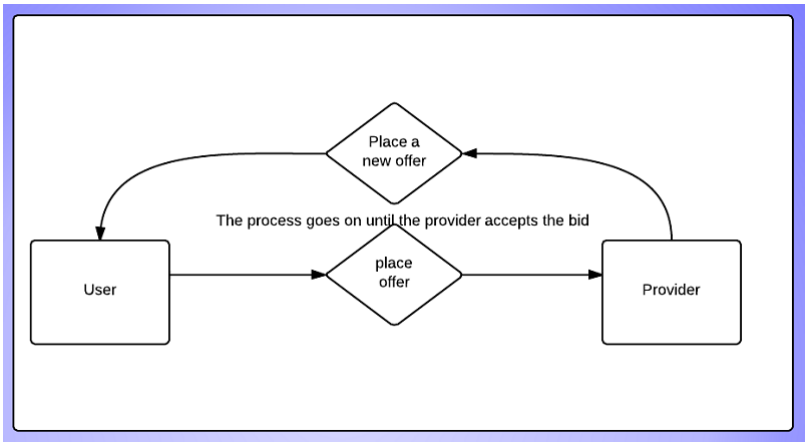

Figure 9: The Figure displays the new negotiation model using the Yankee auction.

set a low price and a maximum price based on 1% and 100% profit. The low price has been chosen in such a way that the provider will not have any loss. In the same way, the maximum price is chosen so the customer does not have to pay so much even if the provider has 100% profit. These are just some assumptions used for the purpose of this dissertation. The consumer then selects the instance that he/she needs to use. After selecting the instance, the user needs to bid on the instance, putting a value that he/she is willing to pay. In case the price bid is lower than the lowest price set by the provider, the user is asked to bid another time. In the end the consumer pays the price that he/she bid as stated in the Yankee auction. The minimum and maximum price from the provider is calculated every time a user wants to bid on an instance based on how many resources there are left.

In case there are many users that are bidding at the same time then the Vickrey algorithm is applied as described in Figure 10. Then the price is set for all the users based on the second highest price bid from all the users. The user can accept the offer or reject it depending on how much he/she is willing to pay. If a user does not agree with the offer, then the consumer can make another offer and so on. The algorithm needs to be pareto-optimal and individual rational in order to be optimal. However, there is a time limit set in order for the user to be able to purchase the instance. No one will actually choose negotiation if they need to wait for too long. So the estimated time to wait and negotiate for an instance is set to no longer than 5 minutes. The time has been set to a specific one because some users require the instances straight away and they choose to go with someone else if the auction goes for too long.

The model described above is going to be pareto-efficient and individual rational because they follow the Vickrey auction. The Vickrey algorithm is pareto-optimal and individual rational. It chooses the second highest bid and not the third one because the

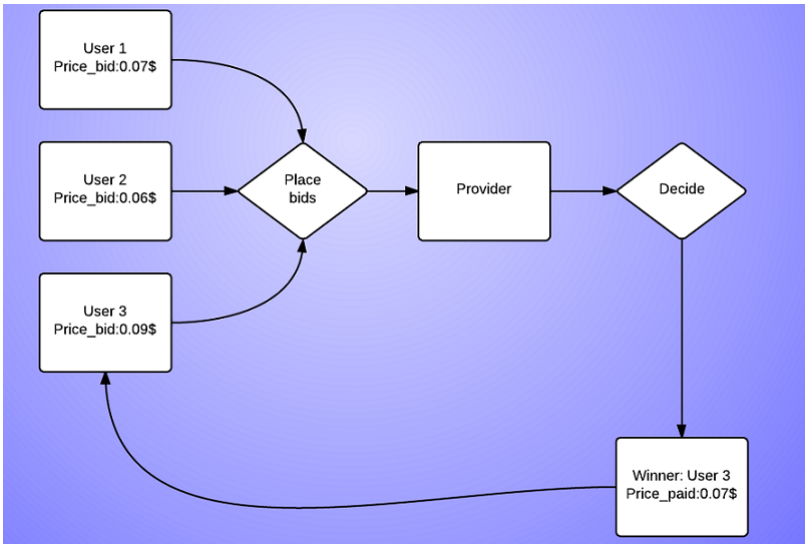

Figure 10: The Figure displays the new negotiation model using the Vickrey algorithm.

first one is pareto-optimal, meaning that there is no further gain from the improvements in resource allocation. We have assumed that since the price is pareto-optimal there is no other option that can benefit both the provider and the consumer at the same time. Since the algorithm is individual rationale as well it means that neither parties can lie about their bids. This ensures a fair trade between the providers and the consumers.

3.5 No-Contract Pricing

Up to now, we have described what is cloud computing, the pricing models offered today, a brief introduction about negotiation and deployed a new negotiation model that handles interaction. So in this section we are going to take a look at a different pricing model that has not been implemented in the cloud yet.

After doing some research about the pricing schemes offered by most of the providers, the conclusion was that all of them provide either on-demand or contract pricing plans. Hence, cloud computing industry can be related to the mobile industry since both of them are trying to attract more users, make profit and keep the business going. The two industries are similar since both of them provide plans for pay-as-you-go and contract plans which vary based on the target users they aim for[36]. Just as mobile providers make their pricing known without any plans, cloud computing follows the same principle having various on-demand prices[37] for different types of instances as mentioned above in Section 2. On the other hand, they provide contract plans in order to give more benefits and subsequently to increase the number of consumers. This is

ensured based on the "service profit chain hypothesis", mentioned in [38], which states that customer satisfaction brings customer loyalty and ultimately increases profitability. Having customers that are loyal to the company brings higher profit through "enhanced revenues, reduced costs to acquire customers, lower customer-price sensitivity, and decreased costs to serve customers familiar with a firms service delivery system"[38]. Moreover, it has been researched that more than 120 mobile companies have 33% profit for the gross margin resulting in the prices being much higher than what they should be[41]. In case the customer becomes unsatisfied with the services, the provider should make a better offer that cannot be refused. This is essentially a benefit for the provider since it has a lower price to keep a costumer than to try to get a new one[39]. Mobile providers, use contract plans to constrain the consumer to stay with a mobile carrier for at least the period of the contract. People usually do not have too much money to spend up front for a mobile phone so the idea of going into a contract and paying a lot less for the phones sounds very appealing to the consumer at first sight. However, they do not realize that by going into a contract they spend much more than what they would normally pay, in case they would have just bought the phone from a retailer[40] and go on a no-contract plan. Similarly, cloud providers, such as Microsoft and Amazon,offer contract plans for 6-months, 1-year and 3-years. For example, an Amazon consumer using a 1-year contract plan for an m1.medium instance using the high utilization pricing is paying 583.28$ for the whole year. From the total amount paid, only 245.28$ is spend on the instance the other 338$ is for the provider to reserve at all times an instance for them. This is equivalent with using the normal on-demand pricing for almost 203 days. However, if the user is aware that they need more than that, they should go with the contract since they do not have any other options. The contractor ensures the consumers have reserved the instances at any time, just as the mobile providers are ensuring capacity for the contract consumer first of all[41]. So by using a contract plan the user has to pay an up-front cost which is more than the resources requested, like mobile providers which end up paying more for the phone because of the additional charges that are coming with the plan. The mobile carriers contract plans are equivalent with the cloud reserved plans. They both require an up-front cost, one case for a new phone and in the other case to reserve the instance. In addition to this the user is charged for the conversations/resources used.

For this reason, mobile companies have recently started providing no-contract plans which has been proved to be profitable, cheaper and preferred by customers because they get more benefits out of it[42]. This has started as being a plan for people with low-income but soon became used by everyone[43]. By no-contract plan, a user gets a certain number of minutes, texts and possibly data included. The plans can be for a day, a week or a month. After the user finishes the data from the plan they are charged at the normal price. In a no-contract plan the users have to get a phone from a retailer or use the one they already have. In case of buying a new phone, they need to pay the

cost of the phone which might be expensive to have just at the start. However, this saves lots of money in the end because there is no activation fee in the beginning. They can choose to use the plan which best fits them. For example, if a consumer only needs more than 500 minutes for 4 months a year, then he/she can only switch for a fitted plan during those months. In all the other months, they can use a cheaper one that suits their needs. By using a no-contract plan the user has the ability to choose different plans each month or switch the provider without paying any termination fee because they are not in a contract. One more advantage that this brings them, is having a phone that can be used with any provider regardless of where they have bought the phone from. Another benefit brought by the no-contract plan is having control over how much people spend[42]. In early years, the contract plans were more used by the consumers, however, now more people are satisfied with no-contract plans that are starting to multiply more, to wrap around the necessity of the users[44].

As a result of the research described above, cloud computing was found to be similar with mobile plans. However, after checking providers, such as Amazon, Microsoft, ProfitBrick, CloudSigma; none of them offer no-contract plans for users. Providers attract much more users by giving them a certain amount of resources for a specified period of time with no up-front cost. The first thing to do in order to try to come up with an offer is to check which types of instances are most used by the consumers. After coming up with a list of most used instances, providers can check which is the average number of resources used per week and per month. This means that those are the instances more likely to be used in the future.

In this section, the assumption made is that the most used instance is m1.medium from Amazon. We have also assumed that the instance is used regularly for 10 hours a day. Earlier in this section we discussed about the year long plan which we have concluded to be 583.28$ for a heavy utilization compared to on-demand which is 1051.2$. We can see that even with the up-front cost the contract plan is almost half the price of the on-demand per year. The price for the instance in a year long contract is 0.066$. Let's say our system has a high utilization for the m1.medium, using the information from above we can come up with some no-contract plans. Assuming that the only cost for the provider is to pay for energy and maintenance from time to time, the lowest price per instance should just include that. This is going to be harder to assume since they do not provide any indication of this. Taking a look at the Amazon m1.medium instance, for the on-demand the price is 0.12$, for a high utilization contract the price is 0.028$ without the upfront cost and 0.066$ with the upfront cost, while a spot instance price can get as low as 0.013$. The no-contract plans should be targeted at two categories, one for the simple users and the other one for companies.

The first plan created was for a simple user that only requires the instances for general use. The plan in this case should contain 10, 100 and 500 resources for a month

assuming that the user does not require more than 6 hours a day. For example, if the user would require to use this at the on-demand prices they are going to spend 216$, 2160$ and 10800$ respectively. Let's assume the instances is given at half of the on-demand price which is 0.06$. This saves the user half of the amount spent for on-demand pricing, and it is not going to tie them to a contract. In case, they were not satisfied with the services after one month they can switch to a different provider. The price was set based on the information from above. Knowing that the price can get as low as 0.013$ and as high as 0.066$, we can choose a price between the boundaries. Ultimately, the providers are trying to have all their resources in use at any time as well as increasing the number of users. Having many more requests they can get more resources for the ones that are over-utilized in order to comply with the users necessities. Moreover, there are users that just want to run an application or build something on a cloud. Due to this perspective, the provider also needs to provide a no-contract plan for a day or a week. This is more likely to be used by smaller users rather than companies because usually the companies require much more computation to be done. The plan for a day is covering 8 full hours at any time. The number of hours have been chosen based on the fact that a person works only 8 hours a day. The assumption made for the price in this case is based on the mobile industry which has 33% profit for the on-demand pricing. Knowing this, the price will be set in order to only bring 10% profit in which case the price will be set at 0.0924$. The plan for a day will be for 1, 10 and 100 resources respectively having a cost of 0.739$, 7.39$ and 73.9$. The plan for the week will contain 35 hours to be used at any time for 10, 100 and 500 resources. The customer will be charged 30.23$, 302.3$ and 1617$ respectively(using the same price as above).

The second plan was created for companies that require much more resources than a normal user. Some companies may only need to make computations for only three months rather than 1-year or 6-months. For this reason, a one-month plan needs to be created for companies that require a higher number of resources. The Amazon website[37] gives more discount to volumes of resources between [3500..8500], deducting that consumers mostly use between [500..3500]. The assumption made is that the provider gives less discount for intervals where they have more users and more discount to where there are only a couple of consumers using it. The price will be set as the one above, designed for a normal user(0.06$). The plan will be fixed for a number of resources such as 1000, 3000, 5000. The no-contract plan will also be set for 15 days worth of hours that can be used at any time during the month or 30 days worth of hours in case of heavy utilization. The plan containing 15 days worth of hours is priced at 21600$, 64800$ and 108000$ respectively. In addition, the other plan for the full month worth of hours is prices at 43200$, 129300$ and 216000$.

In every single plan, mentioned above, the user can save money. At the same time the consumer is not tied up for a full year contract or more. The only thing left for the consumer to do is to make sure they are using the correct no-contract plan that suits their

Normal User		
Period	Resources	Cost
Day - 8 hours	1	0.739$
	10	7.38$
	100	73.8$
Week - 35 hours	1	30.23$
	10	302.3$
	100	1617$
Month - 180 hours	10	108$
	100	180$
	500	5400$
Company		
Month - 360 hours	1000	21600$
	3000	64800$
	5000	10800$
Month - 720 hours	1000	43200$
	3000	129300$
	5000	216000$

Table 8: The table shows the no-contract plans for cloud that will be implemented later.

needs. When the plan runs out they are charged the on-demand price like in the mobile industry. As seen above, the plan is for normal users and for companies, however, this does not mean that they are fixed. So a company can use the plans from normal users and vice-versa. It all depends on what each of them needs the resources for.

4 Design

We have looked at how the new negotiation model and the no-contract plans are going to work. In this section, we design the mechanism for implementing a fixed pricing model, the negotiation model and the no-contract plans described above. Throughout this section the lowest price refer to the price containing only the energy amount as specified in Section 2. The provider in this case does not have any profit nor a loss. The project has a front service which contains the command line interface accessed by the users as well as a back-end service that contains the database. The database keeps a record of the users with the amount they paid and the amount they currently have to pay and a record of the instances provided. The instances in the database provide a more complex view because every time a user requests an amount the program checks whether that instance in the database has that specific amount available. By keeping a

database the user is also going to be advised whether the instance requested exists or not. The user has only read access to the last bill he/she has to pay, as well as the prices for the instances and the instances available. The database set-up can be seen in Figure 11.

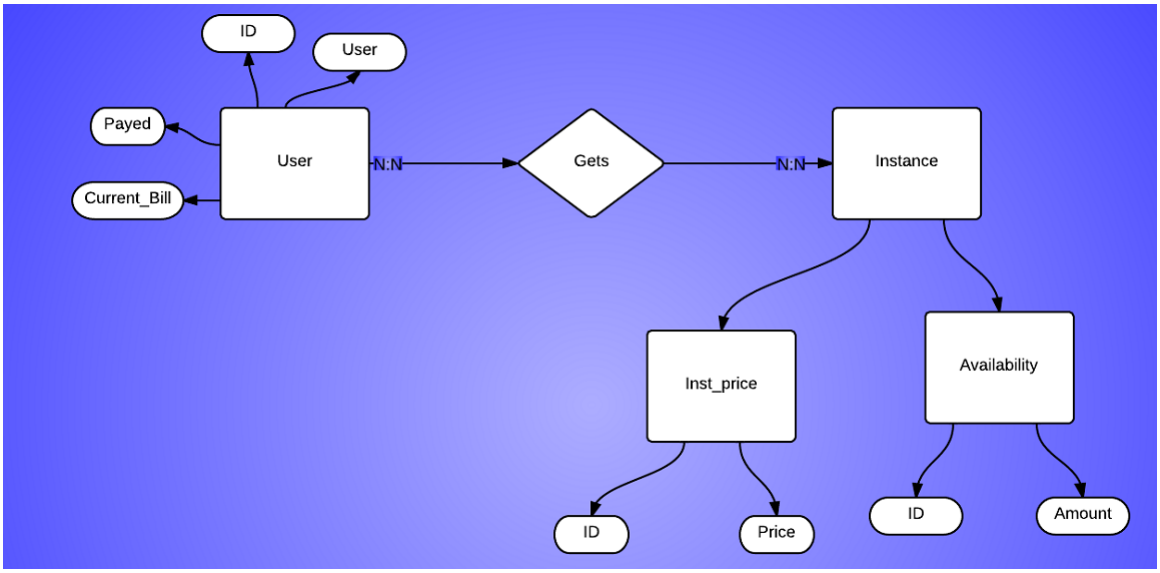

Figure 11: The pictures shows the entity model of the database.

4.1 Fixed Pricing Model

The fixed pricing model is one of the simplest models that cloud computing provides as mentioned in the sections above. Cloud computing provides on-demand pricing also known as fixed pricing. Each instance has different architectures which results in a different price for each of them based on the complexity of the hardware. Moreover, the energy price differs based on the different types of instances. In this dissertation, our fixed price contains the maintenance cost(the lowest cost) plus 33% profit, as it is in the mobile industry.

For the fixed price model the mechanism is straight forward. The user can check the types of instances, the price per instance as well as choosing an instance if it was not already selected through the command line. For the first option, the consumer can check the types of instances offered by the provider. Each containing details about the architecture. After choosing the type of instance, the consumer can check the price. The API handles the fixed pricing demand based on the variables passed through the command line. Having decided all this, the consumer can choose not to use the provider or

go ahead and reserve the instance(s). The consumer can then choose the amount needed from that instance. The provider checks if they can provide the users demand. If the system cannot provide that instance for the amount requested the user gets some options: to wait until the provider can provide the right amount, choose to go for a different type of instance or end the process. Considering the option where the provider has the amount of resources requested, the consumer is going to be allocated the instances for the time specified in the command line. When allocating the instances, the back-end server, containing the instances available should be updated and the charging should be started. This is done in parallel in case there are more consumers accessing the amount at the same time(for this software we will be using threads). The *Fixed_Pricing Class* implements a thread so the updates can be done one at a time using mutual exclusion. The consumer keeps the instances until he/she chooses to terminate them. When the instances are terminated the user is charged for the period used and the instances are made available again. The consumer is able to make a rough estimated of the bill based on how many hours the instance was used but they do not have access to the bill until the instances have been terminated. Once the user receives the bill, the database is also updated and the user can check the current amount that he/she needs to pay. If the provider does not have the resources requested it prompts the user to see whether he/she wants to use a different instance at the price it is offered or choose to come back later. For the purpose of this design, the only time the users can see what types of instances are available and what price they have is when they choose to opt for the fixed pricing. The fixed pricing method in this dissertation is just a simple way of showing how the pricing model works and become familiar with the things required to build more complex models on top of it such as interactive negotiation. The whole process can be seen in Figure 12.

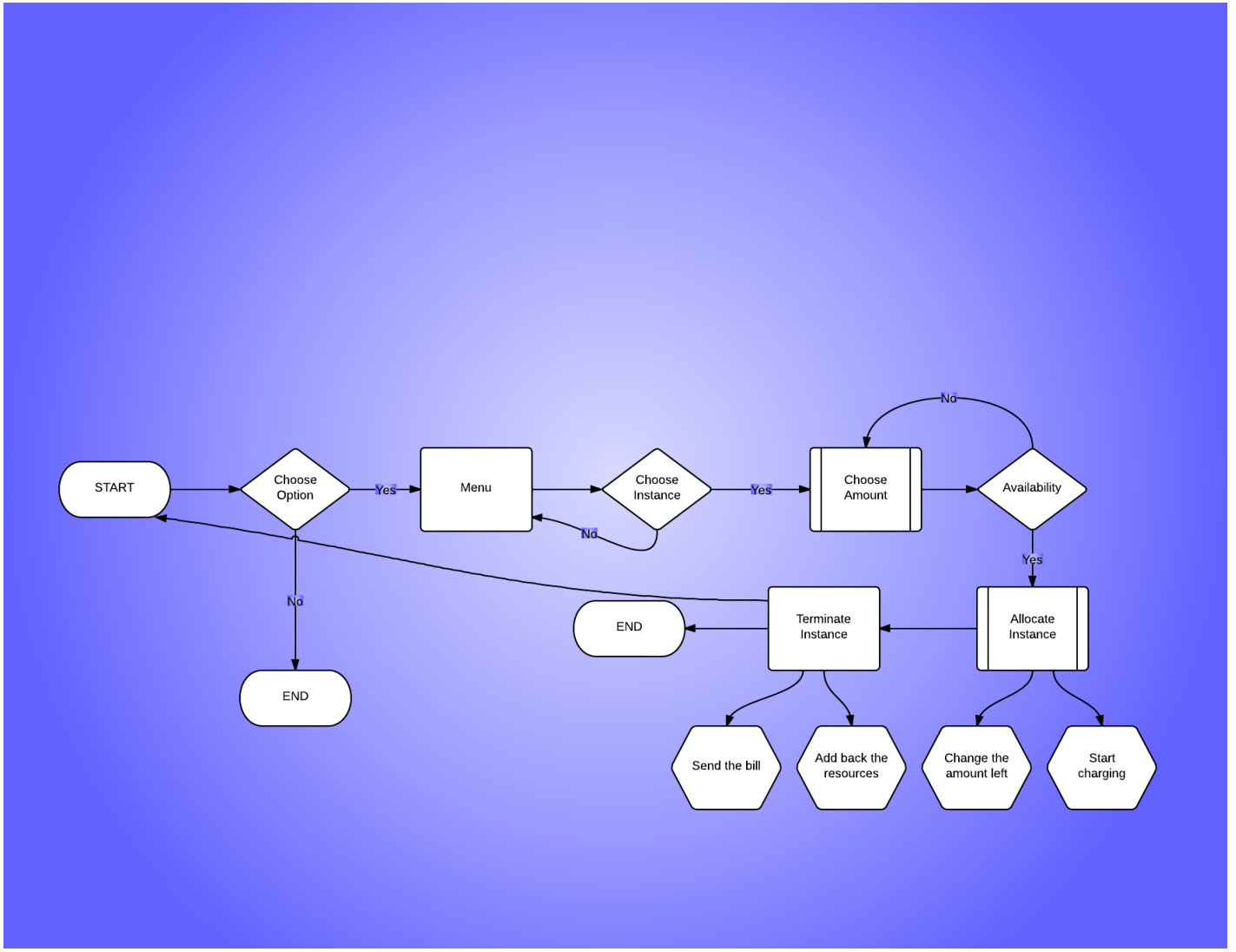

Figure 12: The pictures shows the mechanism of a fixed pricing method.

4.2 Negotiation Pricing Model

This subsection describes the negotiation mechanism proposed in this dissertation. Interactive negotiation has not been introduced yet into the cloud models as mentioned in Section 3. As we have seen above, negotiation can have a huge impact on the provider as well as the consumer since both of them can benefit from it, when the negotiation is closed.

The new mechanism developed is considered to be similar in a way with the Spot Instance Pricing from Amazon. However, it provides a different approach since users can rent resources for unlimited time with a low price. In this API the consumer is requested to choose an instance and specify the necessary amount, as well as the time the instance is going to run for. The time is set before, in order to make it easier to analyse the benefits. After this has been done and the provider has checked that they can provide the users' demand, the consumer can go on to bid the price he/she is willing to pay. When the bidding starts the provider automatically sets the lowest price and the highest price. In this case the highest price is considered to be 100%, the maximum profit the provider can get out of the resources, while still not charging a lot. We have considered this because the provider is going to get profit out and the consumer does not need to pay too much. The price can change, however, based on the number of users at that time. If there are many users and the demand is higher, then the lowest price will be just the maintenance otherwise, it can change. If only one user is bidding at a time then the user needs to bid a price between [lowest_price..higest_price]. Once the price is in that range the provider accepts it and the resources are allocated to the consumer by paying the price he/she bid. This is developed after the Yankee auction, because the user pays for the price he/she bids. The other option is when there are multiple consumers bidding for the same resource at the same time. In this situation, the Vickrey mechanism is going to be applied. The users are all blindly bidding the amount willing to pay and the provider is choosing the highest one. If there are more consumers with the same highest price then it is going to choose the one with the highest quantity. However, the consumer pays the second highest bid rather than the one it bid like in the Yankee auction. Another case to consider is when we have multiple users and multiple auctions happening at the same time in which case the Vickrey is applied for each auction. The others are withdrawn from the auction and they can start bidding again. The bidding can only go for 5 minutes in order to balance the time so the user does not have to wait very long to use the instances. The 5 minutes interval has been considered as a balance so the user does not have to wait too long to get the instances requested. The time limit is just a block of time that we have considered to be sensible based on the fact that users do not want to spend too much time when they need the instances right away. When a user receives an offer in which both parties are satisfied, the resources are allocated and the same billing process as in the section above applies. However, the user might

never win in the 5 minutes interval in which case, he/she needs to keep trying until their bid gets accepted. After the resources have been allocated the databases are updated accordingly. The process should give the user the possibility to achieve a lower price in the end. This is analysed, later on, in order to see whether it provides benefits for both the user and the provider. The whole process described above is illustrated in Figure 13.

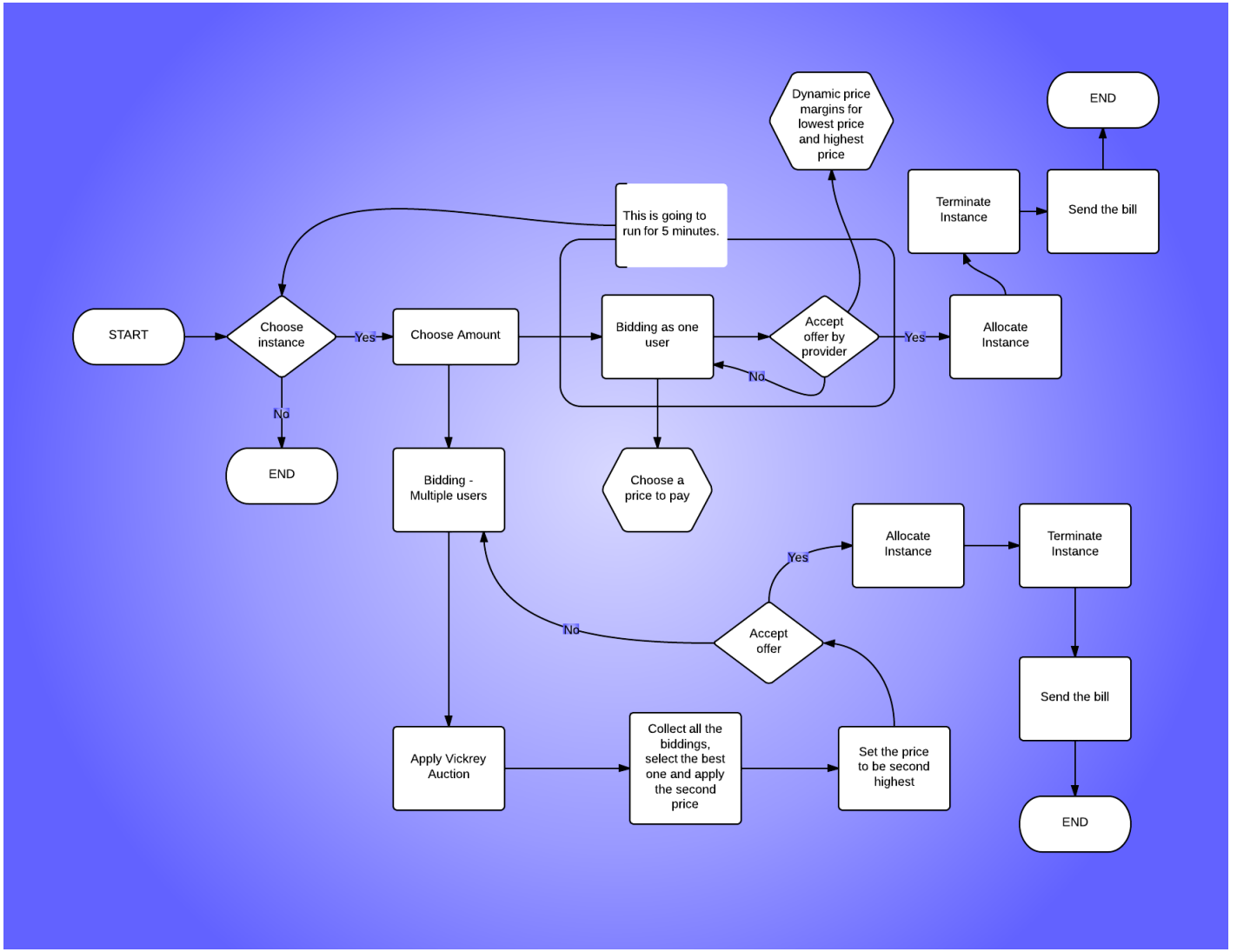

Figure 13: The pictures shows the negotiation mechanism proposed in this dissertation.

4.3 No-Contract Plan Pricing Model

Up to now, we have talked about the fixed pricing scheme and the negotiation design, now we are going to introduce the mechanism for a no-contract plan. As discussed in the sections above, mobile carriers have adopted this type of pre-paid charging. For users this is an alternative for not paying the up-front cost required to get a mobile phone. The same applies for a cloud computing provider, the only difference being that users pay an up-front cost to reserve the instances.

This model contains a couple of packages that users can rent for a day, a week or a month with no upfront cost. The packages have been set in Section 3.5. The user decides which package best fits what he/she needs to do, keeping in mind that whatever goes over the plan is paid at the on-demand price. Once the user decides the plan, the provider has to check whether the plan is still available. We considered that the provider has a number of limited packages for each plan in order to be able to provide other pricing schemes. After choosing the plan, it allocates it to the consumer until it runs out. A new table is going to be included into the database to keep count of how much the user has left from his/her allowance. The user is not allowed to have another plan until the period has elapsed. The API gets updated because only a part of the resources are allocated for the no-contract plans. Once the plan has expired, the user loses any hours that he/she has left from the plan. For this pricing model, we considered that the user has to pay when choosing the plan and not after like in the other cases. The server is updated with the number of plans left form each category. The process is described in Figure 14.

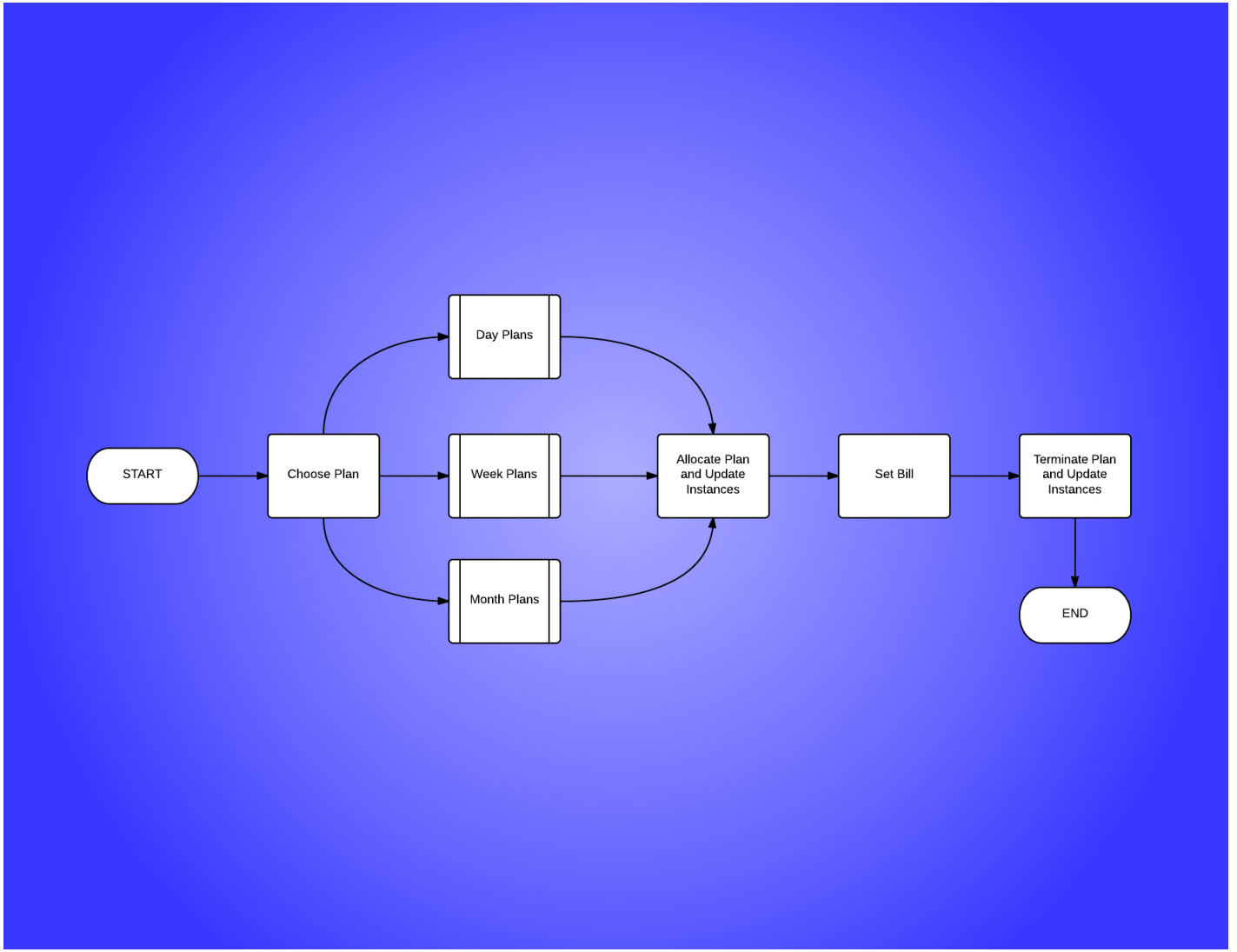

Figure 14: The pictures shows the no-contract API model.

5 Implementation

Previously, we have created a design that helps building the implementation and structuring the dissertation. This section gives an elaborate understanding about how the implementation has been developed as well as showing that the output is accurate. First, the reader gets an insight of how to start the program using the command line menu, followed by the description of the fixed priced implementation, and continuing with the implementation of the negotiation. The negotiation mechanism is of two types: for a user and multiple users. This is based on the parameters send through the command line.

5.1 Command Line Menu

The program comes with a command line in order to make it easier for the user. The user specifies in the command line what his/her requirements are. The command line handles multiple users who can choose to go for the fixed pricing or the negotiation mechanism. The user will have to specify the time required for the job to run in order to be able to test the price. Every variable is going to be checked to see if it satisfies the conditions or the boundaries. Each user has a *user_id* and a *user_name*. As mentioned in Section 4, a database has been created to hold on the information about the users and the instances. The database contains the amount paid by the user up to that point plus the current bill. The user has to pass its *user_name* every time he/she runs an instance. The program has multiple arguments containing different users that are executed at the same time. Each argument contains the parameters required to start a pricing model as follows:

python main.py [-u username [-d [-p]]][-u username -s model -i instance_id -a amount -t time]

where:

- -u, –user - login with the user requested. The user name needs to be unique. Allows the same user to pick a negotiation and a fixed pricing to be executed at the same time.
- -s, –scheme - refers to the pricing model the user wants to select. The user can put *f* for the fixed pricing scheme, *n* for the negotiation scheme and *nc* for the no-contract plans.
- -i, –instance - specifies the instance id that the user is requesting. In case the id is invalid it will ask the user for another instance id.

- -a, –amount - contains the amount the user is requesting from a specific instance. If the amount is not available, the user can take the amount available plus the remaining from another instance or choose to pick a different instance.
- -t, –time - for the purpose of this dissertation the user should specify the amount of time that he/she wants to run the program for. This makes it easier to test the pricing model.
- -b, –bid - in case the user has chosen the negotiation model, he/she needs to specify the price that he/she is willing to pay for an instance to run. If there are multiple users, each one of them needs to specify the price they want to pay for the instance.
- -d, –typeOfpricing - a list of all types of instances is provided as we can see in Figure 15. Each instance has a unique *id* assigned to it. Followed by the name of the instance and a brief description of what the instance offers. These are set up by the provider.
- -p, –priceOfInstance - the program provides an option for the users to check the price for the instance that he/she is interested in. A list is going to pop-up with the price for every instance as follows in Figure 16.

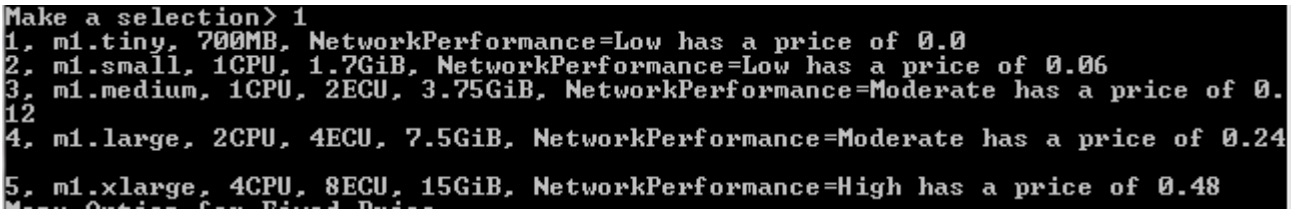

Figure 15: This Figure shows the types of instances that our system will provide. It provides details about each instance as well as the price.

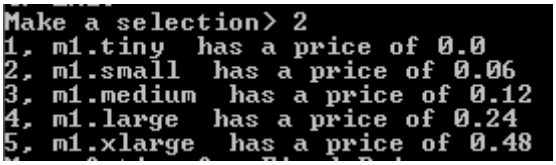

Figure 16: This Figure shows the prices for each instance.

Some of the arguments above are mandatory and the user is not able to call the program without setting them properly. The users are prompted with a message saying that they are missing arguments from the command line. In case they are not sure what is missing, the program provides a *–help* command which gives users the information about the required fields as we can see in the Figure 17.

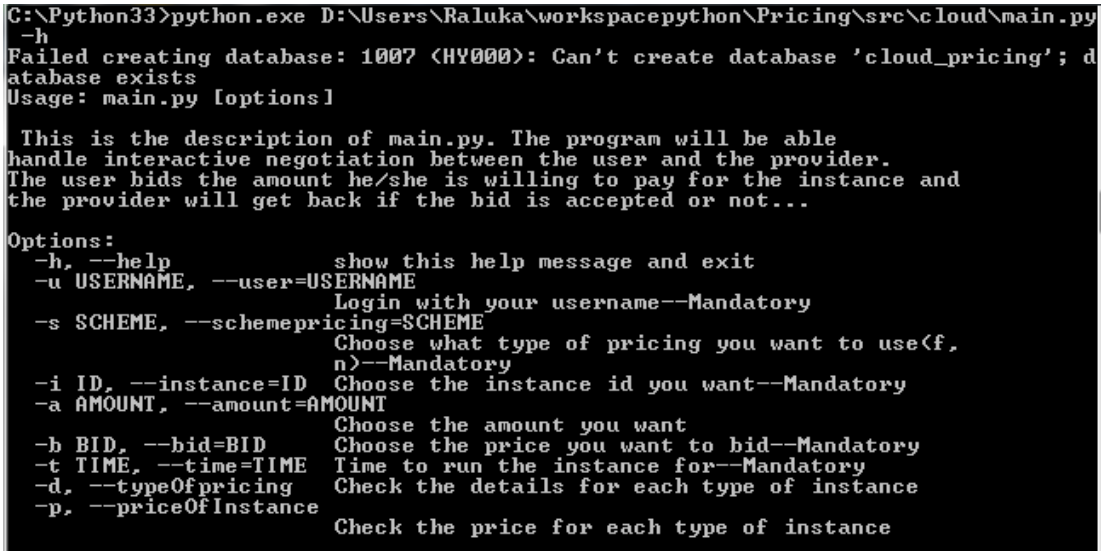

Figure 17: This Figure shows the help menu provided to use when calling *python main.py -h*. In case the user forgets the arguments necessary to run the program, he/she can just access the command.

5.2 Database

The first thing that the implementation requires is a database which will store the *name of instances*, *details about each instance*, *a unique id* in order to be able to select the instance that the user requests, as well as the *price* for each instance. The database is generated through python so the administrator does not need to recreate it manually every time something goes wrong. When we created the database, we have decided to use the Connector/Python module because *mysqldb* module is not supported in Python 3.x. The database for the fixed pricing scheme consists of three tables: instances, availability and price. All tables are developed through python in order to make it easier for the administrator. In case of failure the database and the tables are generated back through running the program. After the database has been deleted the tables are populated and created next time the program is executed. This makes it easier for the administrator because he/she does not require to create the tables by hand every time something goes wrong in the database. Deleting the database can be done as well through python. Every time a new user requests to use the pricing model, the table is updated. When the user runs the command line, each variable is checked in order to see if the conditions are satisfied based on the values from the database. The database contains information about how much the users spend up to that point and what is the current bill. If the user exists already, the database just gets updated. The tables are created using the *CREATE TABLE* command from MySQL. Values are included in the table using *INSERT* com-

mand. When the instances are allocated the availability table needs to be updated using the *UPDATE* command.

5.3 Fixed Pricing Implementation

We have talked about the command line and the database, now we are going to start looking at the fixed price implementation. As mentioned above in Section 2, a fixed pricing scheme is also known as pay-as-you-go. Therefore, the user pays for how long the resources have been used. In our implementation, we have assumed that the user is charged per hour and the system contains a total of 1000 resources for each type. The user can choose to select whichever they think suits the computation best. For the purpose of this dissertation we are going to assume an infinite number of resources. In case the provider cannot accommodate the demand of the user, he/she has to wait until resources are available or choose a different instance. In this implementation, we only provide the first generation standard instances mentioned in Section 2. The reason for this is that these are similar for both Amazon and Microsoft which makes it easier to provide analysis in the next section. In order to have a fixed-pricing scheme we need to set a price for each instance.

After doing some research[45], we came across the fact that an Amazon facility has an estimated cost of $88 million for 8MW. This includes 46000 servers. Amazon has estimated the price of a new server at around 1450$. The monthly cost contains 57% allocated for new servers, 18% for power and cooling, while energy is only 13% out of the total cost of the facility. Knowing this we can calculate how much Amazon spends for energy per month. An Amazon facility uses 8MW per month for 46000 servers. We know that the total cost is $88M and energy is 13% out of that, meaning that 11440000$ is allocated for energy. From that we can calculate how much it is per server which we are interested in. After doing calculations we saw that a server uses 0.173MW from which we can show that a server consumer energy of 43$ per month. This is the lowest price that our instances can have from which the provider cannot have any loss. Moreover, the provider does not have any benefit either. Having all this information we can create now the lowest price per instance which is going to be 0.06$. We calculate this as being the price for a small instance which uses the least amount of energy. In order to calculate the price for the other instances we are going to use the mobile carrier profit margin which we have mentioned in Section 3. The mobile carriers profit margin is 33%. The pricing scheme developed for this part of the implementation has 33% more than the instance before as we can see in Table 9.

After setting the prices that are going to be used along this section, we explain how the program is developed. The first thing a user needs to do is set the fixed price flag in the command line so the program knows what the user requested. The user should

Instance Type	Price
Small	0.06$
Medum	0.08$
Large	0.106$
XLarge	0.14$

Table 9: The table shows our pay-as-you-go pricing based on information from an Amazon data center.

have checked before what type of instance he/she is going to request and what is the price per hour.

The user executes the program with the *instance* that best fits the type of calculations, the *amount* needed and the *time* required to run the instance. In case the provider cannot offer the amount requested, the user has to wait or choose to rent a different instance. The price set is going to be the one of the new instance selected. When accessing the database to check for availability, the provider should ensure that only one user at a time can access the database in order to make sure that no two users overwrite the information in the database or request instances that are actually not available. The program runs for the period of time requested by the user. After terminating the instance, the user is prompted with the bill that he/she is going to pay, depending on the time elapsed. The user is charged per hour according to the price for that specific instance. Then the user can choose to pick another instance or choose to exit the program as seen in Figure 18.

```
C:\Python33>python.exe D:\Users\Raluka\workspacepython\Pricing\src\cloud\main.py
 -u me -s f -i 2 -a 10 -t 61
Failed creating database: 1007 (HY000): Can't create database 'cloud_pricing'; d
atabase exists
THIS IS ON-DEMAND PRICING!!!!!!!!!
Instance has been Allocated Successfully!!!!
Amount due to pay is:  12.0
Do you want to choose different instances?(yes/no)
no
Exit....
```

Figure 18: This Figure shows the process of choosing an instance and running it.

5.4 Negotiation Implementation

Now, we are going to look at the negotiation implementation. The negotiation starts if the user will select the pricing scheme relevant for it. The user needs to specify the n parameter for the pricing scheme when running. This part of the implementation consists of the new model developed that is able to handle interactive negotiation between the user and the provider. For the purpose of this implementation the program

needs to be able to handle concurrent accesses to the database. The program gets the values passed from the command line such as instance id, time, amount, price willing to pay for a specific user. All this information is processed through the *Negotiation Class*. The program checks to see if the user has provided a valid instance id and if not he/she is going to be asked to put in another instance id. After looking for the instance, the program makes sure that it can provide the amount of resources requested.

In this implementation, we looked at two of the cases. In the first case, only one user interacts with the provider. The user places a bid when starting the program. The provider checks if the price is greater than the lower price set by the provider. The lower price and the maximum price are private so the user does not have any knowledge about them. This assumption has been made because if the user is aware of the lowest price he/she is never going to have an interest in paying more than that. In this case the provider is never going to make any profit. The user requires to bid until the provider accepts his/her offer as shown in Figure 19 below. This is the Yankee auction mentioned in Section 3, where the user pays the amount it bid.

```
C:\Python33>python.exe "D:\Users\Raluka\workspacepython\Pricing\src\cloud\main.p
y" -u me2 -s n -i 1 -a 10 -t 60 -b 0.001
Failed creating database: 1007 (HY000): Can't create database 'cloud_pricing'; d
atabase exists
Choose a different price to bid. This has not been acctepted!!!
0.002
Try again...
0.009
Try again...
0.01
Choose a different price to bid. This has not been acctepted!!!
0.012
Instance has been Allocated Successfully!!!!
Amount due to pay is:  0.012
```

Figure 19: This Figure shows the process that a single user is passing when bidding a price acceptable for both the provider and the consumer.

The second case handles multiple users bidding at the same time. As mentioned in Section 2, the models available now do not support negotiation between a seller and multiple buyers. The user can test how the program handles a multiple consumer negotiation by giving more arguments in the command line. If there are multiple users then the program handles them accordingly. The provider gets the bids from all the consumers and decide who the winner is going to be by looking at the highest price. However, the user will pay the second highest price that was bid. All of these are done in the *Multi-Negotiation Class*, which receives a dictionary of all the users negotiations given through the command line. The class has a method which is going through the dictionary and determine the winner and the second highest price that the user needs to pay. When the highest user has been found and the amount for that instance is available,

the user is going to be charged at the second highest price for the amount requested. The rest of the users need to start again either as a one user negotiation or a multiple negotiation between users until their resources get allocated.

```
C:\Python32>python.exe D:\Users\Raluka\workspacepython\Pricing\src\cloud\main.py
 -u me -s n -i 1 -a 10 -t 60 -b 0.02 -u you -s n -i 1 -a 10 -t 60 -b 0.01
Failed creating database: 1007 (HY000): Can't create database 'cloud_pricing'; d
atabase exists
The winner is  me  and the price paid is going to be  0.01
Instances have been allocated!!! All the rest should start again
Amount due to pay is:  0.01
```

Figure 20: This Figure shows the winning user and how much it will pay.

In case, multiple users apply for multiple instances the provider collects the information for each instance and calculates the price beforehand. If the price from the users bidding for the same instance proves to be better than choosing the highest one, the provider is going to allocate the resources to the users that are going to produce the highest price using one instance. For example, if we have 3 users that are bidding for a medium instance and each one of them bids 0.02$, 0.03$ and 0.04$, while 2 other users bid for a small instance with bidding prices of 0.02$ and 0.04$, then the provider will theoretical pick the highest for each one of them meaning 0.04$ and 0.04$, respectively. The user that requested a medium instance has to pay 0.03$ while the user requesting a small instance pays 0.02$ as seen in Figure 21. This gives the provider a total of 0.05$, but if the provider decides to go with all the users that bid for a medium instance there is going to be a profit of 0.04$ as seen in Figure 22.

Figure 23 shows the structure of the implementation in a UML. The program has a *Main Class* which receives the command line and parses it accordingly based on the options provided in the *Parser Class*. The *Main Class* appends every variable from the arguments given in the command line and then looks through the list of pricing schemes to decide which one is FIXED PRICING or NEGOTIATION. The *Fixed_Pricing Class*, *Negotiation Class* and *Multi_Negotiation Class* act as threads. When the *Main Class* finds a user which requested the fixed pricing, it creates an instance of that class

```
C:\Python33>python.exe D:\Users\Raluka\workspacepython\Pricing\src\cloud\main.py
 -u me2 -s n -i 2 -a 10 -t 60 -b 0.02 -u you -s n -i 2 -a 10 -t 60 -b 0.03 -u me
 -s n -i 2 -a 10 -t 60 -b 0.04 -u bla -s n -i 3 -a 10 -t 60 -b 0.02 -u bla2 -s n
 -i 3 -a 10 -t 60 -b 0.04
Failed creating database: 1007 (HY000): Can't create database 'cloud_pricing'; d
atabase exists
This is a multi-user negotiation!!!!!!!!!!
The winner is  me  and the price paid is going to be  0.03
Instances have been allocated!!! All the rest should start again
Amount due to pay is:  0.3
The winner is  bla2  and the price paid is going to be  0.02
Instances have been allocated!!! All the rest should start again
Amount due to pay is:  0.2
```

Figure 21: The figure shows winners of the bids.

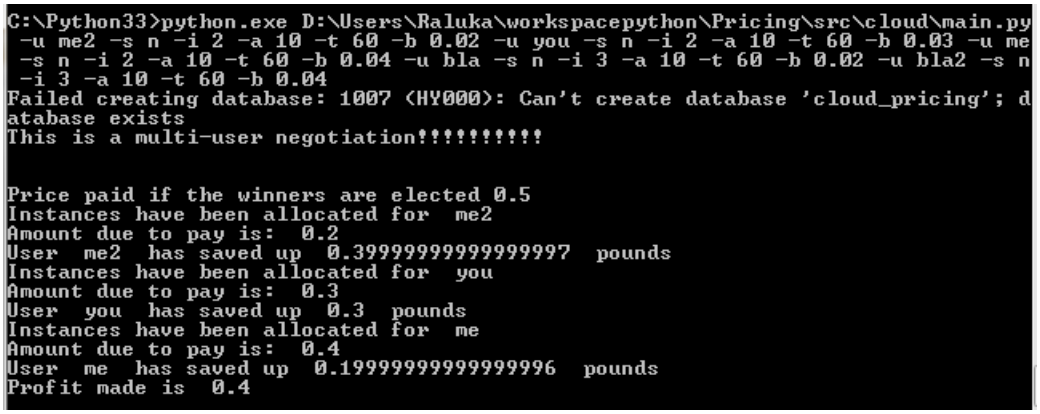

Figure 22: Figure shows the allocation of resources to all users requesting an instance rather than the winners for each of the instances. The profit has been proven to be higher as we can see.

and run the instance with the instance requested and the amount wanted in case it is available. The *Fixed_Pricing Class* starts the running method and checks if the instance exists. If the instance does not exist then it asks the user to choose a different one. After the instance has been selected, it checks the amount requested. If the amount is not available it prompts the user for another amount. If the user has not asked for a fixed pricing, then it goes through the negotiation. As we have seen above the negotiation has two sides, one that handles a single user using the Yankee auction and one that handles multiple ones using the Vickrey auction. The program decides which one the user requested by looking through a dictionary which has stored instances as keys and command(s) as elements in a list. The program deletes the instances from the dictionary that do not match the id's from the database. If the instance has only one element in the list then it creates an instance of the *Negotiation Class*. After doing the same checks as the ones mentioned above, the provider decides whether the bid is between the boundaries. The boundaries refer to the lower price and the maximum price, mentioned in Section 3. In case the price is not between the boundaries, the user is asked to bid again until they get to an agreement and both parties are satisfied. However, if the list has more than one element it gets processed in the *Multi_Negotiation Class*. Here, the Vickrey model is going to be applied. The user that bid the highest price is going to win but he/she is going to pay the second highest price according to the auction model. There is another class that handles all the pricing as well as billing the user. The *Pricing Class* also has a function for checking beforehand whether choosing the winning user is better than giving the instances to multiple users. We have created also a helper class for the functions that access the database to update or insert the values. This has facilitate the programming because they can be accessed from everywhere in the program.

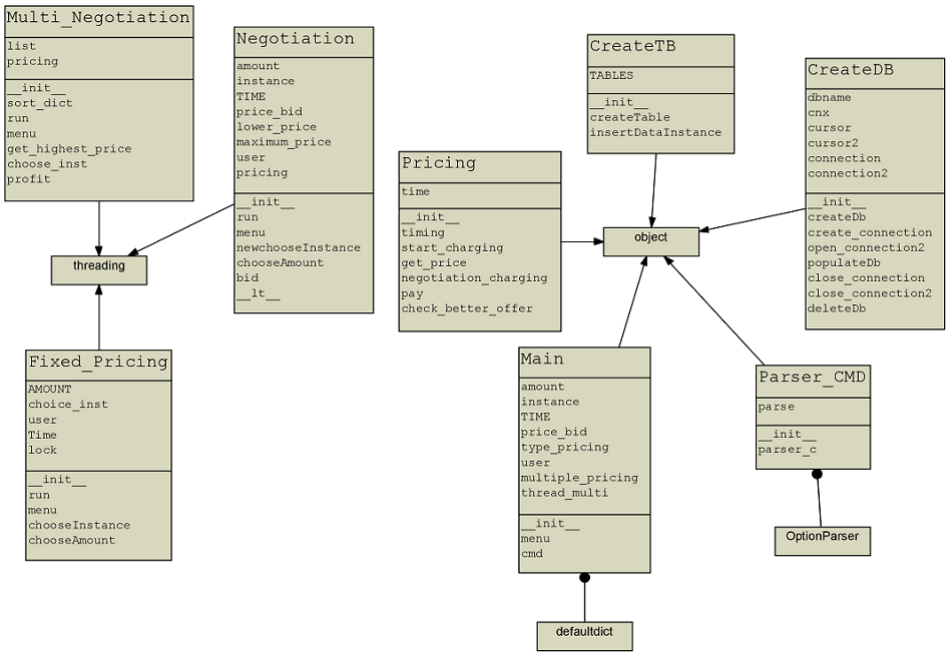

Figure 23: The figure shows the UML diagram of how the program was build.

5.5 Testing for accuracy

In this section, we are going to show that the program provides accurate results. Pricing models play an important factor inside a company. If the provider charges the user more than it should it will destroy the company's reputation.

The first thing we are going to prove is that the program always ensures repeatability. Whenever a program is executed with certain input it always needs to provide the same output no matter how many times we are running the program. This is guaranteed as we can see in Figure 29.

Another test going to be enforced is checking that the pricing is produced correctly. The user is going to pay based on the price bid or price for the instance. The instances, in this case, are going to be charged per hour so when the user goes over one hour and terminates the instance, he/she has to pay for two hours. In this case, we need to ensure that the user is charged accordingly for the right amount of time. Lets take a look at an example to show that it produces the right output. For example, if one user wants to request a medium instance for an hour and 10 minutes, he/she will have to pay 0.12$ since the price for one instance is 0.06$. We can conclude that the program produces the right output as we can see in Figure 24.

We have seen that the price for a fixed price scheme is produced accordingly so

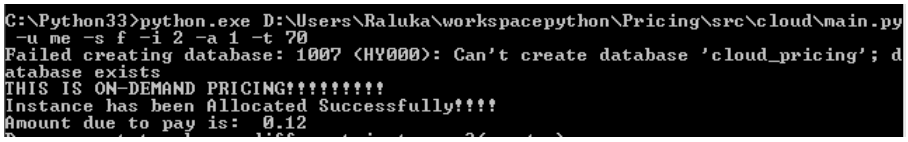

Figure 24: The figure shows that the price for a medium instance that runs for 70 minutes produces the right amount to be paid by the user.

now we are looking at the price for the negotiation mechanism. In this case, we take as an example, 3 users requesting a medium instance at different prices. As we have seen in Section 3, the user with the highest bid is going to win but he/she pays the second highest price. For this we have devised a dictionary that stores a sorted list of all the users bidding for that instance so the program looks at the last 2 elements only and get the information needed. However, the user is going to get the instance requested for the amount requested at the second highest price. The example tested in this case, contains 3 users that have bid 0.02$, 0.03$ and 0.04$ respectively. They all requested the same time and the same amount(10 instances for an hour). The program should pick the user that bid for the highest price which in our case is going to be the user that placed a price of 0.04$. However, the user pays 0.03$ for the instance which is the second highest price. To conclude the program should output 0.3$ as the final answer as we can see as well in Figure 25.

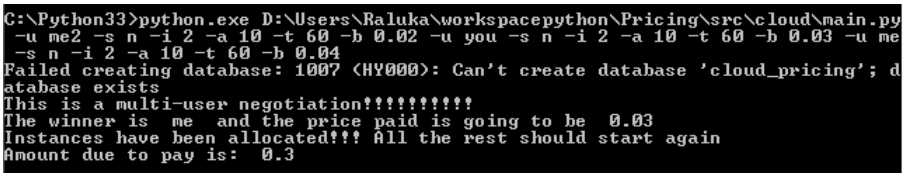

Figure 25: The figure shows that the user that bid the most won the bidding process while the others have to proceed again. The program outputs the correct price the user is going to have to pay for his/her requirements.

The two cases mentioned above, show how the model behaves when the user runs a fixed pricing model or a negotiation pricing model. As we have seen in these cases the output is produced correctly for this two particular cases. However, to run other cases we are only changing the parameters in the command line rather the behaviour of the functions themselves. From here, we can conclude that the program produces the correct output every time we run it for both the fixed pricing model and the negotiation model.

6 Analysis

We have discussed about cloud computing, about the role that the pricing scheme plays, about different negotiation models, then we explained the design and implementation about our new pricing model; which leads us to the part where we are going to analyse the results found in order to come to a conclusion. The analysis is going to show whether the new pricing scheme is beneficial for both the consumer and the provider. By beneficial we refer to the provider trying to maximize the profit and increase the utilization and the user being satisfied with the price. Looking back at Equation 2 we can see that the negotiation can only occur when the utility function is satisfied. The utility function in our model checks whether the conditions from the command line are satisfied, such as the amount. We have decided in Section 3 that we are going to use the desperate strategy, which states that if one sub-negotiation is found then the process is closed which is going to be applied for the one user negotiation process. However, if there are more sub-negotiations coming at the same time the highest one is chosen, which is going to be applied for the multiple user process which implements the Vickrey algorithm. In the end, we have described what properties mentioned in Section 3, does our model support.

We have seen in Section 3.5 that the price to power an instance is 0.06$ based on research done with respect to an Amazon data center. However, the information might not all be accurate about how Amazon handles the data centres because some of it is confidential. Since we do not have any other information, we have assumed that for the purpose of this dissertation we are going to use the same price for each type of the resources while doing the negotiation. The lowest price is going to be set at 0.06$ per hour. As mentioned in Section 3 the maximum price has 100% profit which in our case is going to be 6$. As mentioned above these are going to be the boundaries for the instances. In this section, we are going to analyse two cases. For the first one, we are going to take a look at what happens when the provider has a limited number of resources but enough to run all the instances, while the second case is going to analyse the part where the provider has only limited resources left. There is another case when the provider has an infinite set of resources. However, for this dissertation we will leave it aside since in this case all the users are going to win in order for the provider to maximize the profit.

First case that we considered is when the provider has enough resources available to run the instances of the users that maximizes the profit. Here we are going to show how the negotiation model behaves in this situation. We started by considering a single user that wants to negotiate with the provider for a specific price which describes the Yankee auction. Assuming the provider has the resources available, the user starts bidding the amount he/she is willing to pay for the instance. The user has to run the

command line with the requirements needed, for this analysis we are going to use the following command:

python main.py [-u username -s n -i 2 -a 10 -t 60 -b 0.03]

As we can see above the user requested 10 medium instances for an hour at 0.03$. However, the users' bid is not between the boundaries, in which case the provider is going to ask the user to bid again. The user has then to decide how much more he/she is willing to pay for the resources. The process goes on until the user and the provider agree on the price however, the user is going to pay the amount bid satisfying the Yankees' auction. This process is beneficial because the provider never loses since the minimum price needs to be satisfied. Also if the user reaches the minimum price or over means that the user is satisfied with the price because he/she is willing to pay that much. Therefore, for a negotiation between a user and a provider the outcome will always be beneficial for both of them.

Using the same case as above, we now take a look at how beneficial the process is for a multi user negotiation. In the multi user negotiation, multiple users bid for one or more resources. As we have mentioned above in Section 3, the Vickrey algorithm is going to be applied for a multi user negotiation. Here the user does not need to place a price between the boundaries. It will be more like a spot instance pricing provided by Amazon. To make the analysis more accurate we are going to explain it through an example. We will consider having three users(bob, ra, me) placing a bid for a medium instance and two users(joe, mike) placing a bid for a large instance as follows:

python main.py [-u bob -s n -i 2 -a 10 -t 61 -b 0.08] [-u ra -s n -i 2 -a 10 -t 60 -b 0.09] [-u me -s n -i 2 -a 10 -t 60 -b 0.03] [-u joe -s n -i 3 -a 10 -t 60 -b 0.04] [-u mike -s n -i 3 -a 10 -t 60 -b 0.11]

Applying Vickrey will lead to choosing mike and ra as the winning users. In this case, ra has to pay 0.8$ and mike is going to pay 0.4$ as we can see in Figure 26. However, after doing more analysis we came to the conclusion that the Vickrey model does not maximize the profit of the provider. As we can see in Figure 26, the provider can choose to go with all the consumers requesting a particular instance since that will maximize the profit as we can see in Equation 3. In this case, the Vickrey model will never win because the price paid is going to be the second highest which means that there will always be a price higher than. Therefore, adding the second highest and the highest one will always add up to more profit. Moreover, this leads to choosing the *patient strategy*, mentioned in Section 3, against the *desperate strategy* because the provider is going to wait until everyone has placed the bid to pick the one with the highest profit. If we take a look at this example, we can see that by deciding on all the users that requested a medium instance, the provider will gain 2.8$, while for the large

instance they will only gain 1.5$. Each one of them produces more profit to the provider than the Vickrey model, in which the users are only paying 1.2$. For this reason, the provider always goes with the option to maximize the profit. After doing research we came to the conclusion that the provider will never choose the Vickrey model because the price is always going to be maximized by picking the users that bid the most for one of the resources because it increases their profit.

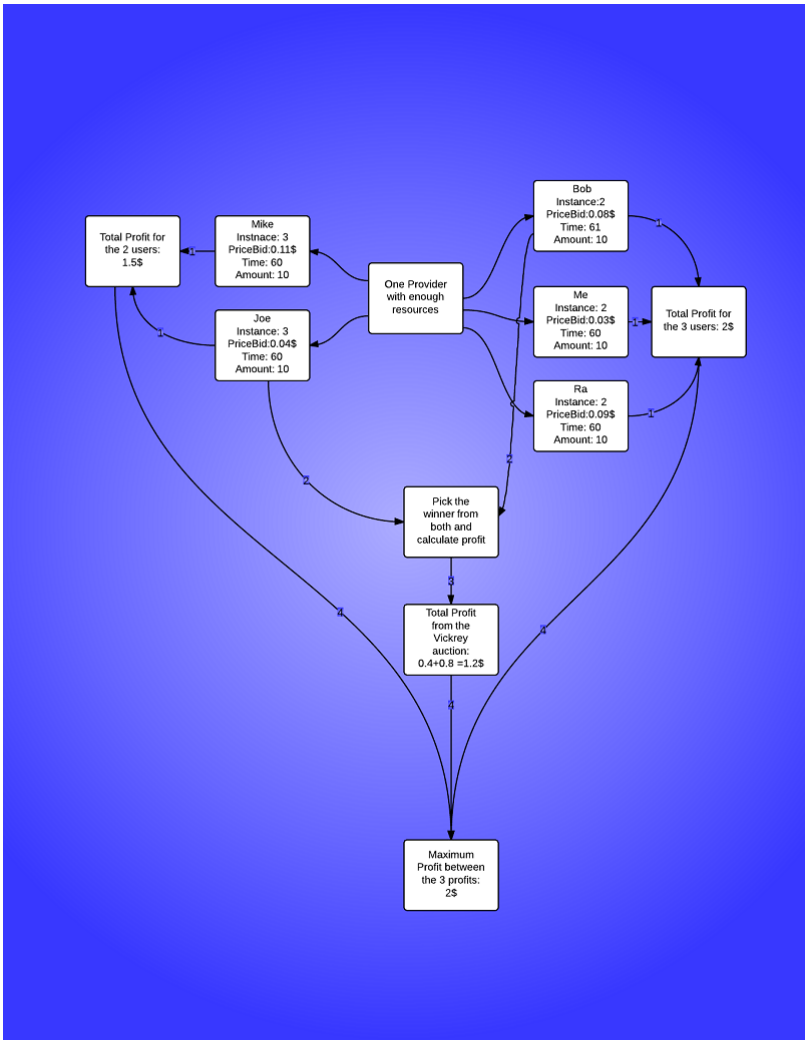

Figure 26: The figure shows the process of negotiation with a multiple users when the provider has enough resources to maximize the profit.

$$max\_profit = max(\sum_{i=winners} price_i * amount_i, \sum_{i=1..n} price_i * amount_i) \quad (3)$$

where:

- max_profit is the maximum profit the provider can get out of the users that bid;
- max provides the maximum between the users that can win with Vickrey or running all the users from an instance;
- price is the bid that each user is willing to pay;
- amount is the number of instances ;
- the first sum refers to the profit the provider makes when running Vickrey;
- the second sum refers to the profit the provider makes when running all the instances of the same type;

The second case we are going to look at is when the provider has only a limited number of resources left. The program running with just a single user is negotiating directly with the provider, if the amount requested by the user is not available he/she is going to be asked to choose a distinct instance as we can see in Figure 27. The user is, however, still going to pay the same price that he/she bid since we have assumed that there is no difference between the instances. The output produces a summary of the instances allocated, the price the user is going to pay and how much he won through negotiation rather than going with an on-demand price.

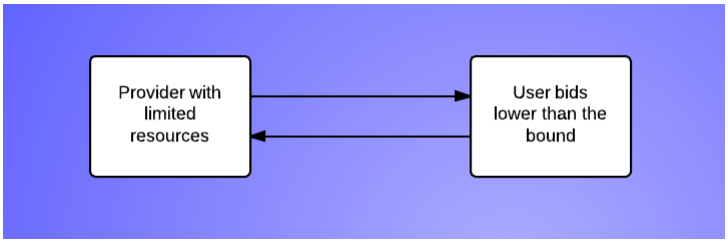

Figure 27: The figure shows the process of negotiation between a single user and a provider based on limited resources.

Now, we are going to take a look at what happens when the provider has resources left for only one user. We are going to use the same command as the one for multiple user negotiation but now the provider has only 10 resources left for a medium instance. The algorithm was created in such a way to offer maximum profit for the provider without unsatisfying the user as described in Figure 28. In this case the provider uses all the amount left and then it asks the remaining users to select another instance. The providers is willing to give a larger instance just for the users participating in the negotiation. In this case the user gets more benefits because it can acquire a better resource at the same price it first bid for the previous instance selected. After running tests we came to the

conclusion that even in this case the user is better off without the Vickrey model. The reason for this is because the provider has more to gain if it gives the resources to the users that brings them more profit. It also satisfies the user because he/she is paying the amount willing to spend for doing the computation.

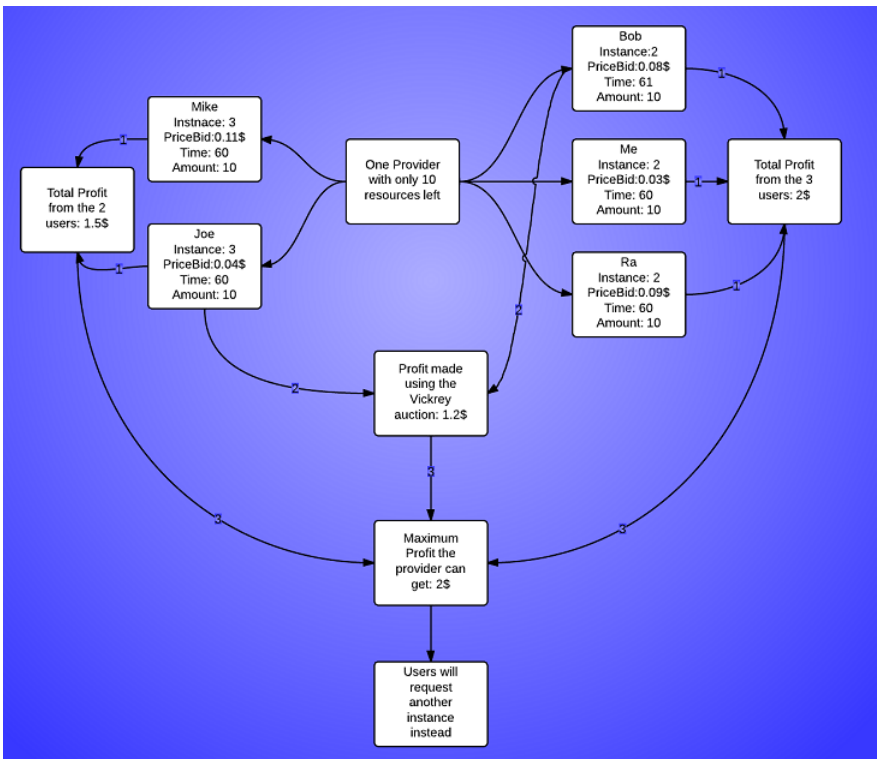

Figure 28: The figure shows the process of negotiation with a multiple users when the provider has only 10 resources left.

Table 10 handles the two case scenarios one where the provider has enough resources(case a) and the other where they have limited resources(case b). In the following part we analysed the results obtained in the table and decided whether the negotiation model can be an improvement for the current pricing models or not. The tables shows two profits for the provider. The *Vickrey* profit is the outcome of the Vickrey mechanism while the *Max_Profit* is the maximum profit the provider can have in each case. The table also mentions the users' satisfaction referring to the satisfaction level of each user based on the pricing model. This will help in deciding whether it is a *good pricing model* or not based on the description given in Section 2.

The first scenario we are going to take a look at is when the provider has enough resources for a single user. As we can see from the table, the provider has no benefit when the user bids below the lower price, in which case he/she needs to increase the

Cases	Initial Start (instance_id, amount, time)	Price Bid	Provider Profit	Resources in Use	Users Satisfaction
a, single user	2, 10, 60	0.03$	n/a	10/10	yes
	2, 10, 60	0.06$	0	10/10	yes
	2, 10, 60	2.4$	40%	10/10	yes
	2, 10, 60	4.2$	70%	10/10	yes
	2, 10, 60	6$	100%	10/10	yes
	2, 10, 60	12$	200%	10/10	yes
b, single user	2-other, 10, 60	0.03$	-	10/0	yes
	2-other, 10, 60	0.06$	0	10/0	yes
	2-other, 10, 60	2.4$	40%	10/0	yes
	2-other, 10, 60	4.2$	70%	10/0	yes
	2-other, 10, 60	6$	100%	10/0	yes
a, multiple user	2, 10, 61	0.08$		30/50	yes
	2, 10, 60	0.09$	Max_Profit:1.6$	30/50	yes
	2, 10, 60	0.03$	Vickrey:1.2$	30/50	yes
	3, 10, 60	0.04$		30/50	no
	3, 10, 60	0.11$		30/50	no
a, multiple user	2, 10, 61	0.8$		30/50	yes
	2, 10, 60	0.9$	Max_Profit:16.63$	30/50	yes
	2, 10, 60	0.003$	Vickrey:8.4$	30/50	yes
	3, 10, 60	0.04$		30/50	no
	3, 10, 60	0.11$		30/50	no
a, multiple user	2, 10, 61	0.8$		20/50	no
	2, 10, 60	0.9$	Max_Profit:24.2$	20/50	no
	2, 10, 60	0.003$	Vickrey:9.2$	20/50	no
	3, 10, 60	0.04$		20/50	yes
	3, 10, 121	1.1$		20/50	yes
b, multiple user	2, 10, 61	0.08$		30/10	yes
	2-other, 10, 60	0.09$	Max_Profit:1.6$	30/10	yes
	2-other, 10, 60	0.03$	Vickrey:1.2$	30/10	yes
	3, 10, 60	0.04$		30/10	no
	3, 10, 60	0.11$		30/10	no
a, multiple user	2, 10, 61	0.8$		30/10	yes
	2-other, 10, 60	0.9$	Max_Profit:16.63$	30/10	yes
	2-other, 10, 60	0.003$	Vickrey:8.4$	30/10	yes
	3, 10, 60	0.04$		30/10	no
	3, 10, 60	0.11$		30/10	no

Table 10: The table shows the cases the program will produce based on the values given.

bidding pricing in order to win. If the user bids the lowest price then both parties are satisfied and an agreement is enforced. If the user bids above the lower price as we can see in the table, such as 40%, 70%, respectively 100% profit for the provider, they both get to an agreement and the negotiation is closed. The provider is going to accept the bid as long as it does not provide any loss for them, which is the case where the user's bid provides 200% profit to the provider. As long as the users agree to bid for a higher price that means they are satisfied otherwise they can just opt to get out of the negotiation.

The second scenario that we are going to analyse is the negotiation mechanism where the provider does not have any resources left from that specific instance requested. This case does not differ too much from the one above because the user can choose another instance to run instead of the one selected. The price he/she wants to pay is staying the same. This is probably not happening in a real life situation. However, since we do not have too much information about the instances we have assumed that all of them require the same amount of energy.

After analysing the table above for the entries specified we can conclude that both the user and the provider are satisfied if the bidding price is over the LB(lower bound price). If the price is under LB then the user is satisfied but the provider is not. Also as the user increases the price he/she is less satisfied, while the provider increases the level of satisfaction. The cases where the user bids closer to LB is the most satisfactory negotiation. We can conclude from here that the negotiation for a single user does not provide an optimal solution for both parties. In order to be optimal, the provider will try to increase the price bid as long as it can while the consumer will try to lower it as much as possible. For this reason, there will always be a better option for one of them. However, they can get to an agreement to satisfy both.

The last analysis conducted handles the negotiation between multiple users. Here we have again the case where the provider has enough resources and limited resources(in this example only 10 left). Both cases produce the same output because the user can choose to go for a different instance if the provider does not have the one requested while still paying the same price. Having discussed this, we can now take a look at the results obtained. From the results we can conclude that the provider is always going to try to maximize the profit as we can see in Equation 3 and in some cases the utilization as well. The user tries to maximize utilization if it produces the highest profit. When maximizing the profit not all the users get to run the instances since the provider does not offer unlimited resources in which case some users are unsatisfied and it might take them longer to run their instances. In terms of a multi user negotiation, the provider tries to satisfy all the users that are going to maximize the profit. However, as we can see in the table not all of them are satisfied in which case those users have to start the whole process again.

As we can see above the program has produced the output that satisfies both

parties through interactive negotiation. After doing the analysis, we can decide whether what negotiation properties the model satisfies. The negotiation mechanism guarantees success because the provider tries to satisfy the user in order to attract more consumers. The model also ensures maximum social welfare which in the case of cloud computing refers to maximizing the number of users. In this model, the sum of the utility function is going to be maximized as we have seen in the analysis above. The algorithm was first devised using the Vickrey algorithm because this was pareto-optimal. However, after analysis we have discovered that the Vickrey algorithm is not pareto-optimal because the provider can choose differently in order to maximize the profit and the utilization. Individual rationale is another property ensured through the model because the users and the provider are playing fair. The next characteristic satisfied is stability in which case the provider cannot perform better than the user. We can see this in the case where the user bids for the lower price which implies the statement before. The algorithm does not offer simplicity because in some cases it is harder to get to an optimal solution and still benefit both parties such as in the negotiation between one user and the provider. To conclude, the algorithm satisfies most of the negotiation properties and satisfies both parties in most of the cases.

7 Conclusion

In this dissertation, we have shown that negotiation is one important factor in making a business grow and have a better reputation. By allowing the user to have a say in the price that he/she is willing to pay for an item, the user is going to be more satisfied while the provider will automatically increase the utilization. We have seen in the analysis that the negotiation model is going to be beneficial for cloud computing pricing models. There can be done more investigation towards other models that might prove to be better. We have used in this dissertation, the auction model Vickrey which is a sealed bid where the user bids blindly for the instances he/she needs as well as the Yankee auction where the user pays the amount it bid.

To close the assumptions made in the previous sections, we have found some cases in which both the users and the providers are satisfied at the same time. However, not all of them are also optimal. For a single user, if the price bid is between the boundaries or over the higher bound both of them are satisfied. In terms of a multi user negotiation, only the users that are allowed to run the instances are satisfied, while the provider is satisfied if their profit and utilization are maximized. After doing analysis we have concluded that the Vickrey algorithm is not beneficial for the provider. Even though, we thought it maximizes the profit we have proven that by choosing all the users running the same instance, the profit is much higher. We have also concluded that the

patient strategy is better than the *desperate strategy* because it will wait until all the sub-negotiations have been done to pick the one that produces the highest profit. However, it has one drawback because time is not important in this strategy but it might be for a user that needs to run the instance very fast.

In this dissertation, we have also conducted analysis for the pricing per minute introduced recently by Microsoft Azure. After doing the experiments, we came to the conclusion that users requiring many resources will definitely benefit from the pricing per minute, while for a normal user the difference is so small that it will be at the user's latitude what provider to choose. As mentioned in Section 2.5.2, Microsoft has stated that the smaller the unit, the more benefits it brings. For this reason, we have analysed how the pricing per second behaves with respect to the pricing per hour from Amazon. The conclusion we came to was that the pricing per second is not worth implementing because it is not better than the pricing per minute. In this case the provider's benefit is going to be so small that it is not worth the hassle.

In conclusion, a negotiation model can definitely have an impact on the cloud computing business. The negotiation can maximize the profit as well as the utilization of the resources. As we have seen above the negotiation algorithm satisfies most of the consumers while increasing the provider's profit and utilization which is definitely a good start. The models can be implemented on top of the pricing models already existing without too many changes.

The project can be further improved. One of the things the project can add is the implementation of the no-contract plan described in Section 3.5. Analysis can be done for this type of plan to see whether it will have the same benefits as the plans provided by the mobile carriers. If the no-contract plans prove to be beneficial, they will have a huge impact for the cloud as they did in the mobile industry. Another improvement is building a GUI instead of the command line implemented now. This will facilitate the user to navigate and test the pricing scheme. The provider can have an option on the GUI to see the information from the database. Through the GUI we can produce graphs automatically, instead of collecting the information manually. People find it easier to visualize rather than looking at lines of output that might not represent anything to a person who is not aware of what the software does. Last but not least, the project can have a back-end server running at all times where each user can send a signal with the specific requirements to the programs' instance. In this way, the pricing can be calculated dynamically. The back-end server will simulate more accurately the pricing model provided by any provider.

A Challenges

Throughout the development of the dissertation we have faced some challenges. One of the challenges that we faced was installing OpenStack. The original idea was to develop the pricing model directly on the cloud. After doing research of what open source cloud to use, we came to the conclusion that we are going to use OpenStack. At first, OpenStack looked easy to use and install. However, after trying to install it for two weeks, we saw that it was not so easy. The installation was rather complicated and time consuming. We have installed OpenStack on top of an Amazon virtual cloud which was running as a front-end server while having other instances being the nodes. OpenStack was developed under Linux 12.04 and the Amazon instance is able to launch a machine and connect to that instance. After doing more research we came to the conclusion that the documentation does not say anything about how to develop a pricing model which made it difficult to progress. This is why we have decided to build a normal software that can just be put on a cloud because it was developed under Python. The program developed just needs to be moved onto a cloud.

Another challenge that we faced was not being able to run it on a cplab machine because it requires some libraries that need to be installed for which we required permission. The program will be able to run on a cplab machine by installing connector/python. This package is required to create the database.

Last but not least, we have tried to get information from Microsoft and Amazon regarding their cloud pricing models but we did not get any information due to confidentiality reasons. we have asked the following questions:

- How does your companies pricing model work? Does the price change based on how many resources are currently free for that specific type of instance that the customer is requesting?
- How easy it is for the company to implement a negotiation mechanism and how easy will it be for the customer to use? Are you thinking in the near future to have a negotiation plan?
- In case where one user wants to request most or all of the resources from within a region because he is doing large computations in a big company, can the user get a better price since he is using most or all of the resources available at that current time in a day? Also what happens if they request all for a very long period of time, such as a month or more?
- Does the algorithm support changes such as handling dynamic pricing or implementing negotiation? And if so, why doesn't it have it?
- How did you come up with this pricing model such as what aspects did you decide

to take into consideration when designing it? Were the aspects from the customer point of view or of the company?

- Why are the users charged by hour and not per day or per minute?
- How do you ensure that the usage is accounted for correctly ? As well as from the business point of view?
- Are users mostly interested in acquiring resources per hour or having long contracts which have an up front cost?
- How do you ensure the pricing -scheme for under-utilization resources?
- What happens when a user requests more resources than what you have available?
- What is the utilization ratio?
- For Amazon, are a lot of users interested in using Spot Pricing? Do people tend to risk having their resources terminated?

B Code

The code bellow shows the creation of the tables though python, as well as how the program connects to the database every time updates need to be done. The code also contains a function that calculates the price for negotiation, as well as a function that updates the database according to the changes made.

```
self.TABLES['users'] = (
 "CREATE TABLE `users` ("
 "   `u_id` int(11) NOT NULL AUTO_INCREMENT,"
 "   `user` varchar(50) NOT NULL,"
 "   `payed` float(11) NOT NULL,"
 "   `current_bill` float(11) NOT NULL,"
 "   PRIMARY KEY (`u_id`), UNIQUE KEY `user` (`user`)"
 ") ENGINE=InnoDB")
self.TABLES['instances'] = (
 "CREATE TABLE `instances` ("
 "   `id` int(11) NOT NULL AUTO_INCREMENT,"
 "   `type_name` varchar(14) NOT NULL,"
 "   `details` varchar(100) NOT NULL,"
 "   `price` float(11) NOT NULL,"
 "   PRIMARY KEY (`id`)"
 ") ENGINE=InnoDB")
```

```
self.TABLES['availability'] = (
 "CREATE TABLE `availability` ("
 "  `id` int(11) NOT NULL,"
 "  `amount` int(11) NOT NULL,"
 "  PRIMARY KEY (`id`,`amount`), KEY `id` (`id`),"
 "  CONSTRAINT `availability_ibfk_1` FOREIGN KEY (`id`) "
 "     REFERENCES `instances` (`id`) ON DELETE CASCADE"
 ") ENGINE=InnoDB")
self.TABLES['inst_prices'] = (
 "CREATE TABLE `inst_prices` ("
 "  `id` int(11) NOT NULL,"
 "  `price` float(11) NOT NULL,"
 "  PRIMARY KEY (`id`,`price`), KEY `id` (`id`),"
 "  CONSTRAINT `inst_prices_ibfk_1` FOREIGN KEY (`id`) "
 "     REFERENCES `instances` (`id`) ON DELETE CASCADE"
 ") ENGINE=InnoDB")

#Start the connection
def create_connection(self):
    dbinfo = {}
    dbinfo['dbhost'] = "localhost"
    dbinfo['dbname'] = "Cloud_Pricing"
    dbinfo['dbusername'] = "root"
    dbinfo['dbpwd'] = "raluca"
    self.connection = mysql.connector.Connect(
                                    host=dbinfo['dbhost'],
                                    user=dbinfo['dbusername'],
                                    password=dbinfo['dbpwd'],
                                    database=dbinfo['dbname'])
    self.connection.commit()
    self.cursor = self.connection.cursor()

#Pricing for negotiation based on the price bid
def negotiation_charging(self, name, inst, amount, t, price):
    time_now = t
    if time_now%60 == 0:
        price_set = (int(time_now/60)*price)*amount
    else:
        price_set = (int(time_now/60)*price + price)*amount
    return round(price_set, 3)
```

```
#Update the bill and pay
def pay(self, name, price_set):
    print("Amount due to pay is: ", round(price_set, 3))
    helper_functions.updateUserBill(name, price_set)
```

C Figures

The figure shows that the program produces repeatable outputs while running with the same input command.

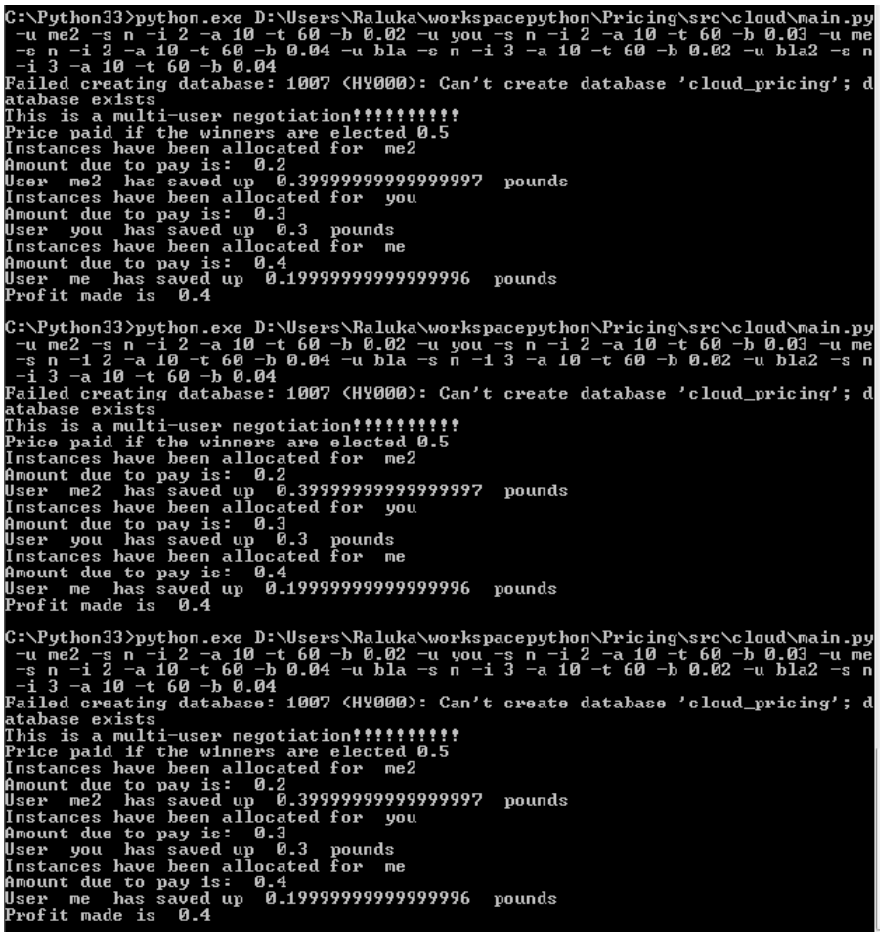

Figure 29: The figure shows that the program produces the same results every time.

The output provided by a multi-user negotiation and the users that are allowed to

run.

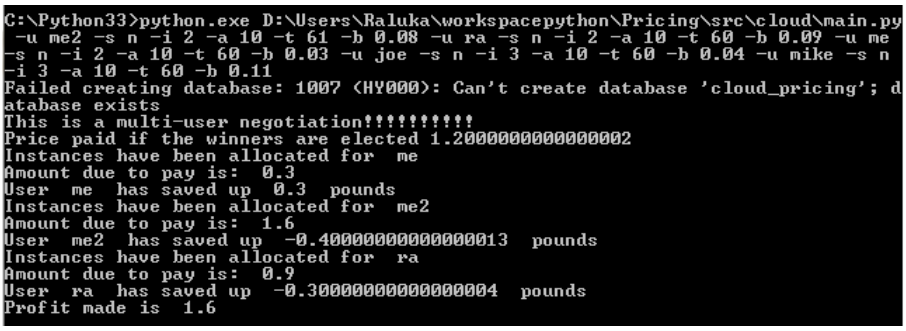

Figure 30: The figure shows the output for a multi-user negotiation.

This Figure shows the output of a simple user that bids below the LB but still continues bidding until they both agree so the negotiation can be closed.

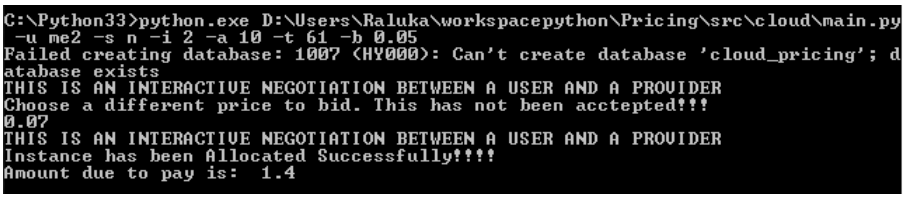

Figure 31: The figure shows the output for a simple negotiation process.

The output for a fixed pricing model that allows at the end to run another instance in case the user needs to do more computations.

References

[1] Becoming the Next Success Story: Using the Cloud to Reap a Profit. *https://spideroak.com/privacypost/business-the-cloud/becoming-the-next-success-story-using-the-cloud-to-reap-a-profit/*(Accessed: August 19, 2013)

[2] Revisiting Cloud Computing: how has it changed - and changed us? Available at *http://www.gizmag.com/revisiting-cloud-computing/26768/* (Accessed: July 8, 2013).

[3] *Introduction to Cloud Computing.* Dialogic.

[4] Michael Armbrust, Armando Fox, Rean GrifĩňĄth, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica,

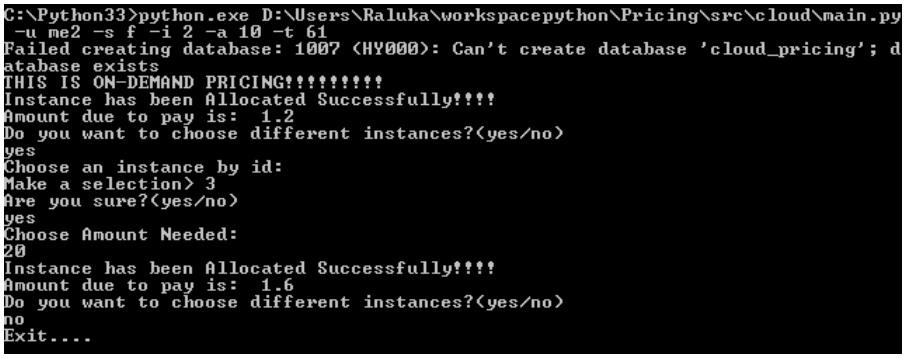

Figure 32: The figure shows a fixed pricing scheme output.

and Matei Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing.* UC Berkeley Reliable Adaptive Distributed Systems Laboratory.

[5] Forrester Report Shows Amazon AWS Reigns Supreme With Developers As Windows Azure Gains Momentum. Alex Williams. Available at *http://techcrunch.com/2012/12/17/forrester-report-shows-amazon-aws-reigns-supreme-with-developers-as-windows-azure-gains-momentum/* (Accessed: June 23, 2013).

[6] Keith Jeffery [ERCIM], Burkhard Neidecker-Lutz [SAP Research]. *The Future of Cloud Computing: Opportunities for European Cloud Computing beyond 2010.* Public Version 1.0.

[7] The Cloud Hits the Mainstream: More than Half of U.S. Businesses Now Use Cloud Computing. Reuven Cohen. Available at *http://www.forbes.com/sites/reuvencohen/2013/04/16/the-cloud-hits-the-mainstream-more-than-half-of-u-s-businesses-now-use-cloud-computing/.*(Accessed: June 24, 2013).

[8] Why Mode to the Cloud? 10 Benefits of Cloud Computing. Salesforce. Available at *http://www.salesforce.com/uk/socialsuccess/cloud-computing/why-move-to-cloud-10-benefits-cloud-computing.jsp* (Accessed: June 24, 2013).

[9] Security and Cloud Best Practices. Aberdeen Group. Available at *http://www.aberdeen.com/aberdeen-library/6846/RA-security-cloud-computing.aspx* (Accessed: July 8, 2013).

[10] Cloud Computing study for Microsoft shows dramatic reduction in carbon emissions. WSP. Available at *http://www.wspenvironmental.com/newsroom/news-2/view/cloud-computing-study-for-microsoft-shows-dramatic-reduction-in-carbon-emissions-235* (Accessed: July 8, 2013).

[11] Giuseppe DiModica, Orazio Tomarchio .*A semanting model to characterize pricing and negotiation schemes of cloud resources.* University of Catania, Catania, Italy, 2012.

[12] Varun Kamra, Kapil SonaWane, Pankaja Alappanavar. *Cloud Computing and its pricing scheme.* Sinhgad Academy of Engineering, Pune, India. 2012.

[13] *Managing the Real Cost of On-demand Enterprise Cloud Services with Chargeback Models.* CISCO.

[14] Andra Raluca(2013). *Pricing Model for Cloud Computing.* University of Edinburgh.

[15] A history of cloud computing. Arif Mohamed. Available at *http://www.computerweekly.com/feature/A-history-of-cloud-computing* (Accessed: June 25, 2013).

[16] Types of Amazon EC2 Instances. The Leading Cloud Operations Optimization Service. Available at *http://www.newvem.com/cloudpedia/types-of-amazon-aws-ec2-instances/* (Accessed: June 25, 2013).

[17] Amazon EC2 Instances. Amazon Web Service. Available at *http://aws.amazon.com/ec2/instance-types/* (Accessed: June 25, 2013).

[18] Amazon lowers AWS pricing again; M3 instances roll out globally. Zack Whittaker. Available at *http://www.zdnet.com/amazon-lowers-aws-pricing-again-m3-instances-roll-out-globally-7000010685/* (Accessed: June 25, 2013).

[19] Windows Azure: Microsoft's Cloud Launches Today. Matthew Weinberger. Available at *http://thevarguy.com/cloud-computing-services-and-business-solutions/windows-azure-microsofts-cloud-launches-today/* (Accessed: June 27, 2013).

[20] What is the new Windows Azure? Jouni Heikniemi. Available at *http://www.redmond-recap.com/2012/06/18/what-is-the-new-windows-azure/* (Accessed: June 27, 2013).

[21] Microsoft Azure Website. *http://www.windowsazure.com/en-us/pricing/details/virtual-machines/* (Accessed August 20, 2013).

[22] Microsoft: FINE! We'll match Amazon - by HIKING cloud prices. Jack Clark. Available at *http://www.theregister.co.uk/2013/04/16/microsoft_azure_ga/* (Accessed: June 27, 2013).

[23] Amazon Web Services: Rise of the utility cloud. Jack Clark. Available at *http://www.zdnet.com/amazon-web-services-rise-of-the-utility-cloud-3040155307/* (Accessed: June 27, 2013).

[24] Microsoft's Azure gets competitive with $1B in revenue. Dara Kerr. Available at *http://news.cnet.com/8301-10805_3-57581994-75/microsofts-azure-gets-competitive-with-$1b-in-revenue/* (Accessed: June 28, 2013).

[25] What is Negotiation? Available at *http://www.the-cost-reduction-consultant.com/whatisnegotiation.html* (Accessed: June 23, 2013).

[26] The Importance of Negotiation in Business. Available at *http://www.the-cost-reduction-consultant.com/ImportanceofNegotiationinBusiness.html*, (Accessed: June, 2013).

[27] Srikumar Venugopal, Xingchen Chu, and Rajkumar Buyya. *Negotiation Mechanism for Advance Resource Reservations using the Alternate Offers Protocol* The University of Melbourne, Australia.

[28] Weiming Shen and Yangsheng Li Hamada H. Ghenniwa and Chun Wang. *Adaptive Negotiation for Agent-Based Grid Computing* National Research Council Canada, University of Western Ontario.

[29] N. R. JENNINGS, P. FARATIN, A. R. LOMUSCIO, S. PARSONS AND M. WOOLDRIDGE, C. SIERRA. *Automated Negotiation: Prospects, Methods and Challenges*.

[30] Raymond Y.K. Lau. *Adaptive Negotiation Agents for E-business*, Department of Information Systems, City University of Hong Kong.

[31] Manoj Kumar and Stuart I. Feldman. *Business negotiations on the Internet*, IBM Research Division.

[32] Iyad Rahwan and Ryszard Kowalczky and Ha Hai Pham. *Intelligent Agents for Automated One-to-Many e-Commerce Negotiation*, University of Melbourne.

[33] Dave Durkee *Why Cloud Computing will never be free*, ACM.

[34] Microsoft will offer Azure by the minute to take on AmazonâĂŹs cloud. Barb Darrow. Available at *http://gigaom.com/2013/06/03/microsoft-will-offer-azure-by-the-minute-in-bid-to-take-on-amazons-cloud/* (Accessed 19th June, 2013).

[35] At long last, Microsoft is ready to compete head on with Amazon Web Services. Barb Darrow. Available at *http://gigaom.com/2013/04/16/at-long-last-microsoft-is-ready-to-compete-head-on-with-amazon-web-services/* (Accessed 19th June, 2013).

[36] Choon Seong Leem, Hyung Sik Suh and Dae Seong Kim. *A classification of mobile business models and its applications*, Yonsei University, Seoul, Korea.

[37] Amazon Web Services. Available at *http://aws.amazon.com/pricing/* (Accessed 21th June, 2013).

[38] Roger Hallowell. *The relationships of costumer satisfaction, customer loyalty, and profitability: an empirical case study*, Harvard Business School, Boston, MA, USA.

[39] Kaj Storbacka, Tore Strandvik and Christian GrÃűnroos. *Managing customer relationships for profit: the Dynamics of Relationship Quality*,The Swedish School of Economics and Business Administration, Helsinki, Finland.

[40] Prepaid or Postpode? The fight for your cell phone dollars. Jessica Dolcourt. Available at *http://www.cnet.com/8301-17918_1-57547193-85/prepaid-or-postpaid-the-fight-for-your-cell-phone-dollars-smartphones-unlocked/* (Accessed 21th June, 2013).

[41] Tommaso M Valletti and Martin Cave. *Competition in UK mobile communications*, Telecommunication Policy, Vol. 22, No. 2, pp.109-131, 1998 .

[42] Prepaid Phones: Same Service, Same Phones, but Less Expensive. Chris McCarthy. Available at *http://voices.yahoo.com/prepaid-phones-same-service-same-phones-but-less-expensive-11990818.html?cat=15* (Accessed 21th June, 2013).

[43] 2-Minute Expert: What are No-Contract Smartphones? Sean Captain. Available at *http://www.technewsdaily.com/3864-2-minute-expert-nocontract-smartphones.html* TechNewsDAily, (Accessed 22th June, 2013).

[44] No-contract options multiply. ConsumerReports. Available at *http://www.consumerreports.org/cro/magazine-archive/2011/january/electronics/best-cell-plans-and-providers/no-contract-cell-phones/index.htm* (Accessed 21th June, 2013).

[45] A Look Inside Amazon's Data Centers. Rich Miller. Available at *http://www.datacenterknowledge.com/archives/2011/06/09/a-look-inside-amazons-data-centers/* (Accessed 10th July, 2013).