A Negotiation-Based Resource Allocation Model in IaaS-Markets

Benedikt Pittl
Faculty of Computer Science
University of Vienna, Austria
benedikt.pittl@univie.ac.at

Werner Mach
Faculty of Computer Science
University of Vienna, Austria
werner.mach@univie.ac.at

Erich Schikuta
Faculty of Computer Science
University of Vienna, Austria
erich.schikuta@univie.ac.at

***Abstract*—Usually, IaaS providers use the inflexible supermarket approach for trading resources: a provider offers a resource for a fixed price. Consumers can buy the offered resources without negotiating with the provider. That's why the supermarket approach is called *take it or leave it* approach. Auctions can be used instead of the supermarket approach.**

Auctions are a way of determining the price of a resource in a dynamic way. Additionally, auctions have well defined rules such as winner and loser determination, time restrictions or minimum price increment. These restrictions are necessary to ensure a fair and transparent resource allocation. However, these rules are limiting flexibility of consumers and providers.

In this paper we introduce a negotiation based resource allocation mechanisms using the offer-counteroffer negotiation protocol paradigm. This allocation mechanisms has similarities to the supermarket approach as consumer and provider are able to communicate directly. On the other hand, the approach has similarities to auctions as the price is determined in a dynamic way. We created a *Bazaar-Extension* for CloudSim which is able to run negotiations. New negotiation strategies and market scenarios can be developed and simulated with the introduced Bazaar-Extension.

I. Introduction

The importance of virtual values chains in enterprises increases dramatically. Virtual value chains as defined by [1] use information to create value. Within the value adding steps of value chains services are utilized. A digital market is the culmination point of stake-holders providing services used along each link in a value chain.

Cloud computing by its characteristic of metered services described by negotiable Service Level Agreements (SLAs) paves the way to the realization of these digital markets [2]. Thus, we introduced a negotiation framework for consumer provider contracting of web services [3] which was referenced in [12] as an example how negotiation can be realised.

In this paper we consider IaaS (Infrastructure as a Service) as an example of a cloud service. Thus we use the terms IaaS provider and cloud provider synonymously.

Usually cloud providers run data centers providing IaaS resources. Providers sell these resources to enterprises in forms of virtual machines (VMs). Today, these machines are mainly traded directly for fixed prices from provider to consumer [4]. We use the term resource for all resources of a provider which can be offered in form of virtual machines.

Virtual machines aren't commodities which are totally interchangeable. Virtual machines are goods which have several characteristics. We characterise a virtual machine using the following descriptors: (i) processing power (ii) storage (iii) RAM (iv) price. The processing power is provided by processors and measured in million instructions per second (MIPS). RAM as well as storage are measured in Mega Bytes (MB).

For our research project focusing on economical aspects of Cloud Computing we needed an adaptive cloud simulation environment which is able to run resource allocation scenarios. In this paper we use the term resource allocation for a mechanism determining how providers and consumers sell and buy resources. We found the simulator CloudSim and three relevant extensions or papers describing how CloudSim was used for resource allocation. (i) The extension introduced by [5] allows to run a double combinatorial auction. It is published on the official CloudSim website (http://www.cloudbus.org/cloudsim/). (ii) The authors of [4] described that they extended CloudSim for running procurement auctions. (iii) The authors of [6] described that they extended the CloudSim VM Allocation Policy by adding a negotiation mechanism. This policy determines which virtual machines are served by which hosts *in* a datacenter (see Appendix A for further information about the VM Allocation Policy). However, we are looking for a marked based resource allocation considering providers and consumers. Further, the presented scenario was simplified by considering storage instead of VMs and neglecting a distinction between Cloud tasks and consumers which makes this work inappropriate for us.

To the best of the authors knowledge no bilateral negotiation exists enabling negotiations and development of negotiation strategies. Therefore, we developed the **Bazaar-Extension** for CloudSim. We developed a negotiation algorithm considering economic theories such as marginal rate of substitution and utility evaluation. Further we incorporated the results of our Cloud Cost Model [7] leading to accurate assessments of cost in data centers. New strategies can be added to the Bazaar-Extension and are currently developed by us.

In section II we describe the basic market mechanism and related works. The negotiation mechanism used in our implementation is described in section III. Section IV describes utility functions of providers and consumers in order to evaluate virtual machines. The counteroffer generation is described in the section V. An implemented negotiation scenario is introduced in section VI. The Bazaar-Extension is summarised in section VII. The paper is closed by a description of further research in section VIII.

For better understanding of the CloudSim framwork we summarized important concepts in appendix A. In the final version of this paper we eliminate the appendix and move it in a technical report as reference.

II. Economical Aspects and Related Work

Digital marketplaces are virtual markets where e.g. virtual machines are traded. Virtual machines can be seen as virtual goods which are supplied by providers and demanded by consumers. So the fundamental market mechanisms for consumer goods can be applied to VMs too. Basically, the market controls price and quantity guided by demand and supply [8]. A typical market model is depicted in figure 1. The numbers 1,2 and 3 refer to three different states to which we will refer in the following paragraph.

Fixed priced services without considering market mechanisms lead to states in which demand and supply differ as shown in figure 1: Setting a high price as shown in state 1 leads to a supply exceeding the demand. We call this state over-provisioning. A low price as shown in state 3 leads to demand exceeding the supply which we call under-provisioning. Both states are inefficient as there is a gap between demand and supply: in both states provider and consumer are not able to sell and buy as many goods they plan at this price. Thus, there is a need for a service-market in order to find the equilibrium where the demand matches the supply as shown in in situation 2 in figure 1. Establishing a market prevents permanent under-provisioning and over-provisioning. Consequently, the market can be considered as a mechanism for finding an adequate price depending on current demand and supply.

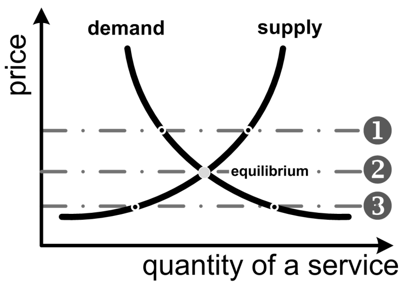


Fig. 1: Basic market mechanism

One of the simplest allocation mechanism is the *supermarket* resource allocation mechanism [10]: Providers create offers which can be bought from consumers. Usually, the offers are not customized. Every consumer buys the same resource for the same fixed price. A consumer chooses the best fitting offer from the providers. The authors of [4] state that a cloud provider usually will not be able to sell all its resources via a such fixed pricing mechanism. It seems like providers have to split their resources into two pools: one pool contains all the already sold resources and another pool contains unsold resources. It is obvious that providers have to sell the unsold resource in order to optimize their profit. Resources which aren't sold by a fixed pricing mechanism are called residual resources [4]. They can be sold using a dynamic pricing mechanism. Dynamic pricing models for residual resources are already business reality as Amazon's spot market [11] shows: consumers can bid for instances. The higher the bid, the higher is the chance of winning an instance. In [4] it is described that the fixed pricing resource allocation mechanism will remain the usual way of selling resources. This is because a provider has loyal consumers with long-term relationships. Both, the consumer as well as the provider profit from such a partnership: providers have a minimum workload and know when and which resources are required by the consumer. Consumers can make long-term decisions as they need not to fear unexpected price increments. Both can control their risk.

A lot of papers describe interesting dynamic pricing resource allocation approaches but most of them are unimplemented in CloudSim. Implementation of different resource allocation mechanism fosters comparability. We see this paper as first step to make different approaches comparable.

Before we introduced the Bazaar-Extension we focused our literature research on implementations for CloudSim which we described in the introduction. The two approaches introduced by [4], [5] are generally appropriate for resource allocation simulation. Both resource allocations are auction based making them convenient for many situations. Nevertheless auctions have weaknesses which we want to emphasize in the following itemization. We want to tackle these issues by introducing a negotiation based resource allocation mechanism realized by the Bazaar-Extension.

- *Constraint 1: Requirements are not fixed*
  Usually consumers need not a specific virtual machine. They may have minimum requirements for the required virtual machine. If they get a better machine for the same or similar price, they will take the better machine. None of the above mentioned mechanisms [4], [5] consider this preference issue as the consumer has to submit a required virtual machine with fixed characteristics.
- *Constraint 2: Mandatory sell/purchase*
  Auction rules are defined to ensure fair competition. Auctions define minimum bids or minimum increments. Further, there may be penalties for the auction initiator if it wants to abort the auction. If for example a provider starts an auction in order to sell a resource, it usually can not abort it even if the environment has changed making the sale unprofitable.
- *Constraint 3: Wait for an auction*
  Usually, auctions have a fixed start-time and end-time. Thus, consumers which want to buy virtual machines or providers which want to sell virtual machines have to wait until an auction starts.
- *Constraint 4: 1:1 situations*
  Auctions are usually used in situations in which providers sell virtual machines to several potential consumers or vice versa. Typically, auctions are inappropriate to handle 1:1 situations. Bilateral negotiations are required for 1:1 situations [10].

III. Negotiation based Resource Allocation

The used negotiation style in the Bazaar-Extension is based on the offer and counteroffer mechanism described in the WS-Agreement Negotiation standard [10].

In a typical negotiation scenario, two negotiation partners such as provider and consumer exchange messages in a alternating way: each negotiation partner waits for offers and responses to them. A simple example for the mechanism is provided in figure 2. A provider starts a negotiation by sending an offer to the consumer at time $t = 0$. The first message is usually called template. Templates are the initial offers published by the providers to promote negotiation [10]. Afterwards the consumer responds to the offer with two counteroffers ($t = 1$). The provider responds to the counteroffers with two new counteroffers ($t = 2$). This message sequence leads to a tree-based structure as figure 2 shows. Theoretically, a negotiation partner can create an arbitrary number of counteroffers in response to received offers. We use the term offer for both, a template as well as a counteroffer. An offer always contains a description of a VM. So we use these terms synonymous.

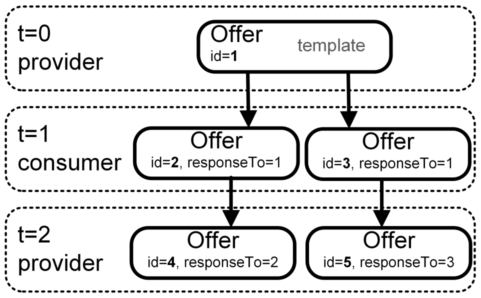


Fig. 2: Offer-Counteroffer example

If both negotiation partners are satisfied with an offer, they form an agreement by exchanging agreement messages. It is also possible to stop a negotiation by sending a reject message. During negotiation, no further negotiation partner is involved. Therefore, negotiations can started at any time with arbitrary templates.

We implemented such a negotiation using the CloudSim framework. We did not implement the whole WS-Agreement Negotiation standard. WS-Agreement standardizes negotiation over the Internet. For example, it standardizes XML-documents containing service agreements and offers. However, CloudSim is a Java based simulation environment where events can be considered as substitutes for XML-messages. Each event is represented by a Java object. We tried to capture the important concepts of the WS-Agreement Negotiation standard and considered them in the Bazaar-Extension.

A negotiation always encompasses exactly one provider and exactly one consumer. However, it is possible to do several negotiations at the same time. If a negotiation partner receives an offer, it has to make several decisions in order to respond to it. In our simulation, consumer and provider exchange three types of messages in addition to the CloudSim messages.

- **Offer**
  Consumer and provider exchange offers which are either templates or counteroffers.
- **Accept**
  If a consumer or a provider considers the offer as suitable it may send an accept request to the negotiation partner which has to send an accept request back in order to form an agreement.
- **Reject**
  If a negotiation party receives a poor offer it can terminate negotiating the received offer. Usually, the negotiation between consumer and provider is continued for other offers. However, there is no obligation to form an agreement.

The criterias determining if an offer is accepted or rejected may change during negotiation.

The WS-Agreement Negotiation standard describes negotiation concepts but no concrete strategy. In the following sections we introduce a summary of a negotiation strategy which we implemented. We do not argue that this strategy is a perfect strategy applicable in all situations. However, we considered basic economic principles. Due to space limitations we focused on the description of the strategy. Advanced evaluations are part of our further research.

Usually the price is the sole decision criterion in auctions. During negotiations different VMs are offered which makes comparability of VMs difficult. A score representing the value of a offered VM and considering all its characteristics including price is necessary to compare VMs and make decisions during negotiation. We created such a score using utility theory described in basic economic literature such as [13].

IV. Utility Evaluation

Utility is a measure of happiness [14]. A typical method for evaluating the utility of an item such as an offer for virtual machines is to use utility functions. In this paper, utility functions represent the utility of an offer for a provider or a consumer. Therefore, offers with high utility values are favoured by consumers and providers. Providers as well as consumers will have different utility functions based on their needs, experiences and resources.

A. Provider Utility

The main goal of a provider is to maximize profit. Precondition for calculating the profit is to define a cost model such as described in [7].

The main contribution of the Cloud Cost Model described in [7] is the distinction between fixed and variable cost which allows to implement more accurate business strategies. Fixed cost are cost which occur independent of the output of the company. The cost which have to be paid even if nothing is produced or offered are fixed cost. Stepped cost such as the administration cost and the broadband cost are fixed cost too. These cost are not increased by each additional unit produced, they are increased by a significant output change, e.g. the administration cost increases with every hundred servers. Variable cost fully dependent on the output quantity. If nothing is produced then there are no variable cost [8].

In our implementation we used the gross margin as utility function for the provider. The gross margin is the difference between turnover and variable cost. So offers which result in a higher gross margin are preferred over offers with a lower gross margin. The following equitation shows the used provider function:

$$U(Provider) = \begin{cases} Price - RAM \cdot Price_{RAM} \\ -Storage \cdot Price_{Storage} \\ -Processing \cdot Price_{Processing} \\ \cdot 1000 \\ \qquad Storage > Max_{Storage} \vee \\ -\infty \text{ if } RAM > Max_{RAM} \vee \\ \qquad Processing > Max_{Processing} \end{cases} \tag{1}$$

The variables $Price_{RAM}$, $Price_{Storage}$, $Price_{Processing}$ represent variable cost. So a received offer for VM has high utility for the provider if either the price is high or few resources are required. The provider has limited resources which are represented by $Max_{Storage}$, Max_{RAM}, $Max_{Processing}$. In our application the result is multiplied by 1000 in order to avoid decimal numbers. Offers violating these ranges are valued with a $-\infty$ utility indicating that it cannot be supplied. Such offers have no value for the provider.

In our further research, we want to enrich this utility functions with further concepts of the previous described Cloud Cost Model. Furthermore, the utility function can be extended by limiting the maximum amount of resources delivered to one consumer in order to enforce diversity. No CloudSim extension we found considers a cost model distinguishing between variable and fixed cost.

B. Consumer Utility

Usually consumers do not require a VM with fixed characteristics. Instead, they have ranges or minimum requirements for the characteristics of the required VM.

In order to calculate the consumer utility of a VM a utility value for each characteristic was calculated: storage, RAM, processing and price. In our implementation, the consumer uses a sqrt-function for storage and a log-function for RAM and processing power. Log and sqrt-functions are typical to represent consumer utility as these functions consider saturation: the utility increase decreases with each further MB for storage and RAM or MIPS for processing power. This saturation reflects real consumer behaviour. A consumer requiring 8 GB RAM will get a great utility increment if it gets a further GB RAM. If 100 GB RAM are offered to the same consumer then the utility increment of a further GB RAM is small. In economics this effect is called diminishing marginal utility [13].

Additionally, the marginal rate of substitution effect of non perfect substitutes such as described in [13] applies to VMs. Storage, RAM, processing power and price are all substitutable to a certain degree. A VM containing 5GB RAM and 1 TB storage may have the identical utility for a consumer as a VM containing 4.5 GB RAM and 1.2 TB storage. However, the less RAM a VM has, the more other resources are needed to compensate further RAM reduction. Thus a consumer which accepts a VM with 4.5 GB RAM and 1.2 TB storage is not willing to a accept further 200 GB storage for a reduction of 0.5 GB RAM. This is because the marginal rate of substitution changes so that the good which is scare gets relatively more worth than the other goods.

The utility function for RAM and processing power is shown in the following equitation. The consumer has minimum requirements of VM characteristics Min_x which are considered in the utility function whereby x is a placeholder for RAM and processing power.

$$U(Consumer_x) = \begin{cases} log(x) & x \geq Min_x \\ -\infty, & x < Min_x \end{cases} \tag{2}$$

The utility function for storage $(sto.)$ is shown in the following equation.

$$U(Consumer_{sto.}) = \begin{cases} sqrt(storage) & storage \geq Min_{storage} \\ -\infty, & storage < Min_{storage} \end{cases} \tag{3}$$

The utility function for the price can be represented with a linear function as shown in the following equation.

$$U(Consumer_{price}) = \begin{cases} Max_{price} - price & price \leq Max_{price} \\ -\infty, & price > Max_{price} \end{cases} \tag{4}$$

In total there are 4 utility functions. In the utility functions (2) and (3) the utility value increases with increasing resources whereby the utility value of the price function in equation (4) decreases with an increasing price.

Using the equations, we calculate the utility of price, storage, RAM and processing power. In order to evaluate the value a virtual machine we need a final utility value. We summarize the utilities of storage, RAM, processing power and price to a total utility value by standardizing them and calculating the weighted sum. The weight indicates the importance of the VM characteristics on the total utility. It is set by the consumer representing preferences.

After the utility of an offer is determined, a negotiation partner is able to decide how to respond to a received offer. Typically, a response results in an accept message, a reject message or a counteroffer. An example how utility functions can be used within negotiation scenarios is described with an Edgeworth-Box.

C. Edgeworth-Box

The consumer is said to be indifferent between two goods or bundle of goods if they provide the exact same utility. Bundles with equal utility form an indifferent curve [14]. Figure 3 shows an Edgeworth-Box visualizing a pure-exchange economy. An Edgeworth-Box illustrates indifferent curves of a buyer and a seller. In the pure-exchange economy illustrated in figure 3 a consumer and a provider (seller) trade cash and HD-Storage. Both the consumer and the provider want to have cash and storage. It is obvious that the negotiation partners want to have as much cash as possible. Further, the provider wants to have as much storage as possible so that it can server other consumers. Consumers use storage to fulfil their business tasks. They may resell unnecessary storage.

The consumer indifferent curves are convex, whereas the provider indifferent curves are approximately linear. Consumers and buyers have an infinite number of indifference curves. The greater the distance of an indifferent curve to the origin the more utility have the cash/storage bundles on that indifferent curve for the buyer/seller. An indifferent curve

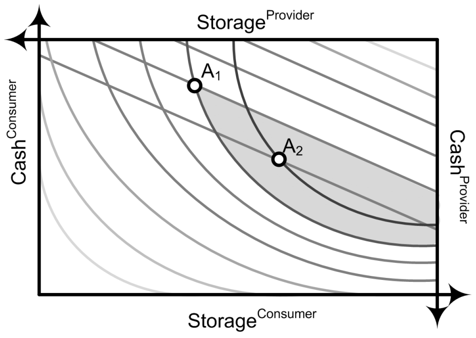


Fig. 3: Edgeworth-Box

which has a lager distance to the origin than another curve is called higher indifferent curve.

Storage as well as cash are limited as shown in the Edgeworth-Box. For example the more cash the consumer gets the less cash the provider gets and vice versa.

In the figure it is assumed that the negotiation partners are currently negotiating and one of the partners has proposed resource allocation A_1. If both, consumer and provider are not willing to accept less cash or less storage then they will not find a better allocation. So they are not able to improve allocation A_1. Figure 3 shows that the allocation A_2 is better for both negotiation partners: In A_2 the consumer and provider reach a higher indifferent curve. Allocation A_2 can be formed if the consumer renounces some cash and the provider renounces some storage. By renouncing goods negotiation partners are able to form a win-win situation. The idea of renouncing a good in order to reach a global optimum is considered in the implemenation description in section V.

The allocation can be further improved until a Pareto optimum solution is reached. A Pareto optimum solution is a situation in which neither the provider nor the consumer can get a higher utility without reducing the utility of the other negotiation partner.

V. Counteroffer Generation

If a offer is received which is neither rejected nor accepted, a counteroffer has to be created. Typical influence factors for counteroffer generation are described in the following itemization.

- **Received Offer**
  A counteroffer should consider the received offer. The received offer is usually an offer which is acceptable for the negotiation partner.
- **Consumer Utility/Provider Utility**
  As already described, utilities are essential for evaluating offers. Counteroffers should have utility for the sending party as well as for the receiving party. So the counteroffer generator has to generate offers which increase utilities of both negotiation partners. Usually, this is a difficult task. However, due to different valuations of virtual machine characteristics, an improvement for both, consumer and provider, is possible as shown in the Edgeworth-Box example.
- **History**
  The history can be used to find out which counteroffers may be successful and which utility values are reachable.

In our initial implementation we consider the received offer as well as the utility of the sending negotiation partner.

A counteroffer should improve the utility of both, consumer and provider. Usually, a greedy counteroffer generation strategy will not be successful. An example of such a greedy counteroffer generation strategy is given in the following paragraph:

A consumer receives an offer for a VM. The received offer is modified by reducing the price and sent back as counteroffer. It is obvious, that the counteroffer is of lower utility for provider than the received offer. Counteroffers which are created by simply reducing price or increasing resource values for RAM, storage or processing power will not be accepted by the provider if it has other potential consumers.

We created two rules for the counteroffer generation:
R1: Modify exactly two VM characteristics of the received offer for creating counteroffers. **R2**: One VM characteristic has to be improved, whereas the other characteristic has to be degraded.
These rules are based on the marginal rate of substitution effect of non perfect substitutes described in section IV-B.

By applying these two rules for counteroffer generation offers can be created which may have higher utility for both negotiation partners.

R1 is used in our simulation to ensure, that the counteroffer remains similar to the proposed offer.

R2 is the basis for finding negotiations results with improvements for both, consumer and provider. The reason for such a win-win situation is the different valuation of VM characteristics. To clarify the valuation differences, we provide the following example.

a) Example: We describe VM characteristics using a vector like $(300GB, 4000\text{MIPS}, 5GB, 10\$)$. The first element represents the storage, the second element the processing power, the third element RAM and the last element the price. In the following we neglect the units in the vector.

We consider that the offer $(300, 4000, 5, 10)$ was received by a consumer. The consumer wishes to have more RAM. So it creates a counteroffer based on the proposed offer: $(300, 4000, 7, 11)$. As you can see, the consumer increased the RAM and price. Nevertheless, the utility of the new offer is higher for the consumer: the consumer is willing to pay more than 1$ for 1GB RAM. However, the consumer offers the provider 1$ for 2GB RAM in the counteroffer.

The provider, who evaluates the offer, may have lower cost than 1$ for the two additional GB RAM. So the new offer is better than the old one for both negotiation partner. This substitution mechanism works for all VM characteristics. Thus, instead of increasing the price, the consumer could try to decrease the processing power in order to get more RAM. It is also possible to decrease for example RAM and storage in order to decrease price. The process of improving VM characteristics and degrading another VM characteristic is called substitution process within this paper.

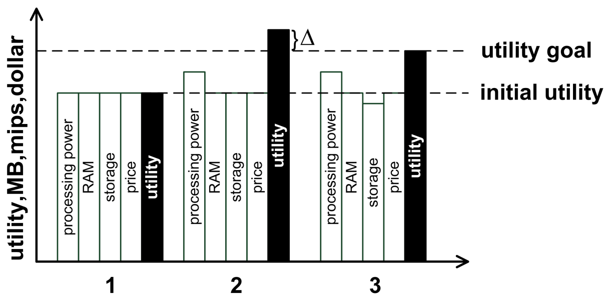


Fig. 4: Substitution process

The substitution process which we implemented in our initial implementation is described in figure 4. The numbers on the abscissa represent the process steps. First the utility of the proposed offer is calculated. Additionally, the VM characteristic which should be improved and the VM characteristic which should be degraded are selected. We simply improve one parameter by deteriorating another parameter. In the example depicted in figure 4 processing power is the VM characteristic which we want to improve. Storage is the VM characteristic which is degraded.

In the second step processing power is increased. The processing power is set to a value so that the utility of the counteroffer reaches the goal utility + Δ. The Δ represents an offset which will be used for degrading a VM characteristic in order to reach the goal utility. In the final step, we reduce the storage until the Δ is compensated. In our example we set storage to a value so that the goal utility is reached. The so generated counteroffer has higher utility for the counteroffer generator than the received counteroffer. Additionally, it may have higher utility for the negotiation partner. There are many ways to reach the goal utility. The presented approach is one of them.

We suggest to keep the Δ low. This is because we want to keep the counteroffer similar to the proposed offer. A big Δ leads to big improvement of processing power while storage is greatly reduced.

Usually, a negotiation partner does not know anything about the utilities of the other negotiation partner. This makes counteroffer generation complex as it is unknown which VM characteristics should be improved and which ones should be degraded. Moreover, the optimal increment size for improving and degrading is unknown. One strategy is to create counteroffers for all combinations as shown in figure 5. This figure shows an initial offer and its counteroffers. The initial offer may be a template offered by a provider. The consumer uses this offer and creates counteroffers by improving as well as degrading all VM characteristics resulting into 12 different offers which have all the same utility for the consumer. A plus indicates that this parameter is set so that the utility is increased whereas a minus indicates that a parameter is set so that the utility is decreased. In the storage panel of figure 5 the storage is reduced, in the processing power panel the processing power is reduced, in the third panel RAM is reduced and in the fourth panel price is increased.

The counteroffer creation process creates 12 counteroffers for each received offer. So the total number of offers exchanged can be calculated using the equation for calculating the sum of geometric series as shown bellow.

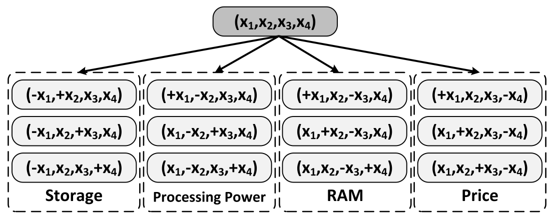


Fig. 5: Counteroffer tree

TABLE I: Abbreviations

Symbol	Description
A	set of attributes $(a_1 \dots a_n)$
a	attribute for describing an offer
a_i	attribute used for increasing $U(o)$
a_d	attribute used for decreasing $U(o)$
o	offer
o_r	received offer
a^o	value of attribute a from offer o
$U(o)$	utility of offer o
R	set of counter offers
i_a	increment steps for attribute a
d_a	decrement steps for attribute a
U_{goal}	utility goal
Δ	offset
U_{accept}^{active}	active utility limit
$U_{accept}^{passive}$	passive utility limit

$$\text{total number of counteroffers} = a_0 \cdot \frac{1 - q^{n+1}}{1 - q} \tag{5}$$

In our counteroffer generation mechanism the quotient q is 12. a_0 is the start value. n is the number of of terms. So a negotiation starting with one offer results into 22621 messages after 4 iterations.

$$\text{total number of counteroffers} = 1 \cdot \frac{1 - 12^{4+1}}{1 - 12} = 22621 \tag{6}$$

This counter offer strategy is formally described in algorithm 1. The abbreviations used within the algorithm are explained in table I.

The counteroffer generation is part of a negotiation strategy. A negotiation strategy decides which offers are rejected, accepted and to which offers a counteroffer is responded. Such a strategy is shown in algorithm 2. The acceptance messages are usual offers which a special flag indicating that they are acceptance messages.

Both consumer and provider have two acceptance utility levels: an active and a passive acceptance level. The active acceptance value is always equal or greater than the passive acceptance level. The difference between these two levels is the following: If a negotiation partner receives an offer which has a utility exceeding the active acceptance level it tries to form an agreement. So it sends an acceptance message. If a

```
Data:
received offer $o_r$
$U_{goal}, \Delta$
$A = \{$processing power, storage, RAM, price$\}$
$R = \{\}$
Result: set of counteroffers $R$
for each $a_d \in A$ do
    for each $a_i \in A$ do
        if $a_i \neq a_d$ then
            /*create copy of offer*/
            $o$=$o_r$;
            while $U(o) < U_{goal} + \Delta$ do
                if $a_i == price$ then
                    $a^o_i$= $a^o_i$-$i_a$;
                else
                    $a^o_i$= $a^o_i + i_a$;
                end
            end
            while $U(o) > U_{goal}$ do
                if $a_d == price$ then
                    $a^o_d$= $a^o_d + d_a$;
                else
                    $a^o_d$= $a^o_d$-$d_a$;
                end
            end
            $R = R \cup o$
        end
    end
end
```
Algorithm 1: Counteroffer creation - consumer view

negotiation partner receives an offer which has a utility which is higher than the passive level but lower than the active level it does not send an agreement message. However, if the an agreement message is received for an offer which has an utility exceeding the passive level the agreement is accepted.

```
Data:
received set of offers O
received set of acceptance request messages M
Result: set of counteroffers, accepts, rejects
sort M by utility descending;
sort O by utility descending;
if $\exists m \in M | U(m) > U^{passive}_{accept}$ then
    send accept acknowledge message for m;
    for $o \in O$ do
        send reject message for o;
    end
    for $r \in M | r \neq m$ do
        send reject message for r;
    end
end
if $\exists o \in O | U(o) > U^{active}_{accept}$ then
    send accept request message for o;
end
for $o \in O | U(o) \leq U^{active}_{accept}$ do
    if U(o)$\leq U_{reject}$ then
        send reject message for o;
    else
        send counter offers for o;
    end
end
```
Algorithm 2: Basic negotiation strategy - high level view

VI. Negotiation Scenario

In this section a scenario is introduced using the concept described above. The results of the scenario show that negotiation leads to higher utilities for both, consumer and provider than the supermarket approach. Additionally, the negotiation result is compared to Pareto optimal solutions.

TABLE II: Simulation Parameters

Parameter	Value
consumer acceptance utility U^{active}_{accept}	7500
consumer acceptance utility $U^{passive}_{accept}$	7500
consumer reject utility U_{reject}	50
provider acceptance utility $U^{passive}_{accept}$	3000
provider acceptance utility U^{active}_{accept}	13000
provider reject utility U_{reject}	50
offset Δ	100
variable cost per MB (storage)	0.0000048828125
variable cost per MB (RAM)	0.00048828125
variable cost per MIPS	0.000075

For the scenario we used one provider and one consumer. The parameters used within the scenario are stated in table II. Both use the utility functions described above for VM evaluation. The consumers minimum virtual machine characteristics are (200,30000,3,100). So if one of these ranges is violated, the VM has no utility for the consumer. The prices form a maximum boundary. So a VM which is more expensive than the price is not accepted by the consumer.

The negotiation is started by the provider which sends one offer-it's template. The provider rejects each counteroffer which has lower utility than the utility of the template.

In our initial scenario, the negotiation is finished as soon as one of the two negotiation partners accepts an offer. An offer is accepted by the consumer if it has a utility greater than 7500. The provider accepts an offer if its utility is greater equal than 3000. The provider tries actively to form an agreement if the utility of an offer is higher than 13000. It has to be considered that the provider and consumer use different utility functions. So their utility values are not comparable.

Generally, for each received offer 12 counteroffers are created as described in a previous section. After 4 iterations thousand of offers exists. So we are not able to depict a complete negotiation tree within this paper.

A pruned negotiation tree is illustrated in figure 6. It shows the path from the initial offer to the accepted offer (agreement). Additionally, the tree visualizes the counteroffer generation mechanism: counteroffers are created by modifying exactly two parameters of the template. For the first counteroffer in $t = 1$ the templates price as well as processing power was modified. The processing power and storage capacity characteristics of the second counteroffer are different from the one of the template.

The initial offer with the characteristics $(300, 4000, 5, 10)$ has a consumer utility of 6299 and a provider utility of 3000. These values are calculated using the utility functions described before. For the provider utility we assumed the following variable cost: (i) cost per MIPS=0.000075 (ii) cost per MB storage=0.0000048828125 (iii) cost per MB ram=0.00048828125 . By using equation (1)the provider utility is 3000 as the following equation shows.

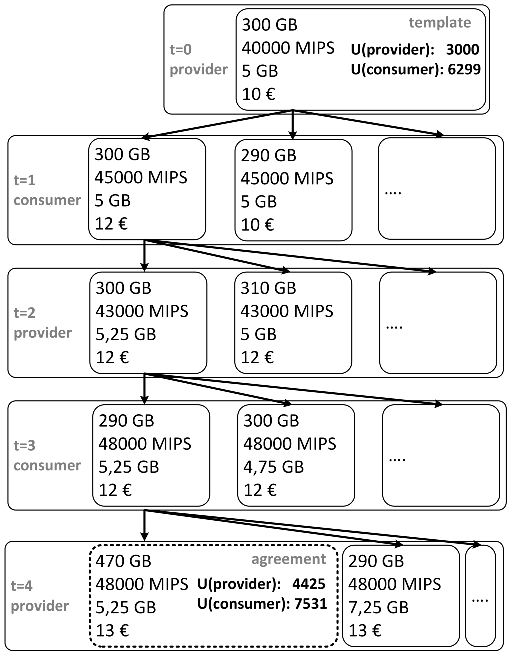


Fig. 6: Negotiation tree from template to agreement

$$U_{provider} = (10 - 300 \cdot 1024 \cdot 0.0000048828125 - 40000 \cdot 0.000075 - 5 \cdot 1024 \cdot 0.00048828125) \cdot 1000 = 3000 \tag{7}$$

In order to calculate the consumer utility the utility factors of storage, RAM, processing power and price are calculated. Therefore, we used the equations (2),(3) and (4).

$$\begin{aligned} U_{RAM} &= log(5 \cdot 1024) = 3.709 \\ U_{processingpower} &= log(40000) = 4.602 \\ U_{storage} &= sqrt(300 \cdot 1024) = 554.256 \\ U_{price} &= 100 - 10 = 90 \end{aligned} \tag{8}$$

Before calculating the weighted sum the utilities have to be standardized. The standardization is done assuming the following market ranges: (i) processing power$_{range}$=30000 to 60000 (ii) $storage_{range}$=204800 to 512000 (iii) RAM_{range}=3072 to 16384 (iv) $Price_{range}$=0 to 100 . Based on these ranges the standardized values were calculated as shown in the following equation.

$$\begin{aligned} &RAM^{stand} = 0.305, \text{processing power}^{stand} = 0.415 \\ &storage^{stand} = 0.387, price^{stand} = 0.9 \end{aligned} \tag{9}$$

The following weights are used reflecting the relative importance of the characteristics: (i) $w_{processingpower}$=0.1 (ii) $w_{storage}$=0.2 (iii) w_{RAM}=0.2 (iv) w_{Price}=0.5

The provider utility can be calculated by multiplying the weight by the standardized value as shown bellow. The result was multiplied by 10000 in order to avoid decimal values.

$$\begin{aligned} &U_{consumer} = (0.305 \cdot 0.2 + 0.415 \cdot 0.1 + \\ &0.387 \cdot 0.2 + 0.9 \cdot 0.5) \cdot 10000 \approx 6299 \end{aligned} \tag{10}$$

TABLE III: Result

	Supermarket	Bazaar	Pareto
U(provider)	3000	4425	≈ 30125
U(consumer)	6299	7531	≈ 8749

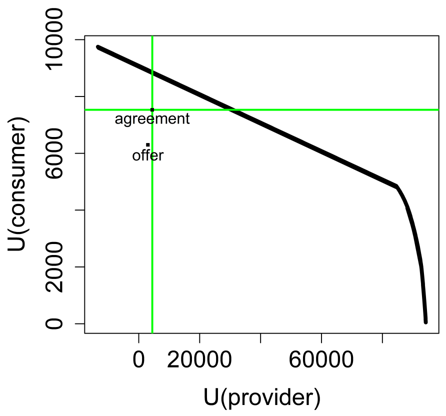


Fig. 7: Pareto border

The accepted offer $(470GB, 4800\text{MIPS}, 5.25GB, 13\$)$ has a consumer utility of 7531 and a provider utility of 4425. So the consumer gets 1232 more utility and the provider gets 1425 more utility.

In a supermarket approach negotiation and consequently improvement does not exist. Thus, if the provider offers the template in a supermarket based allocation process, the consumer can take it or leave it. By negotiation both, consumer and provider are better off.

The results of the negotiation based scenario are summarized in table III.

The Pareto border representing the Pareto optimum is illustrated in figure 7 by the black line. In this figure, the initial offer as well as the agreement are depicted. We calculated the Pareto optimum by an iterative approach: we iterated overall possible service combinations and looked for Pareto optimal points. For offer creation stepwise increments were used for all service characteristics: RAM: 256 MB, Storage: 10 GB, Processing Power: 5000 Mips and Price: 1. So our results are approximations.

Some datapoints of the Pareto border are shown in table IV. Neither the supermarket result, nor the negotiation result is Pareto optimal. However, the negotiation result is nearer to a Pareto optimal solution. The negotiation result can be further improved by increasing the acceptance limits an thus increasing the number of iterations needed to form an agreement.

Figure 7 has an interesting curve shape. The first part of the curve seems to be approximately a straight line while the second part of the curve shape seems not to be a straight line and has a strong negative slope. We tried to explain this curve shape by using the characteristic vectors of Pareto efficient points. The consumer utility of the Pareto optimal characteristic vectors was recalculated without considering the price. The resulting points are called *resource points*. Most of them have a new consumer utility (y-value). Of course, their x-value remains the same. The resource points are shown in figure 8. The points do not form a smooth line. The reason for the non-smooth curve shape is that we used an iterative approach based on stepwise (not continual) modification of the characteristics to get approximated Pareto efficient points.

Based on figure 8 we can conclude that that the resource points remain stable for $U_{provider} <\approx 72000$. So the utility considering storage, RAM and processing power is not decreased. However, the consumer utility considering price is decreasing (see figure 7). So it is obvious that for all points where $U_{provider} <\approx 72000$ the utility decrement is resulted by the price. The Pareto optimal data confirms that the VM characteristics processing power, RAM and storage remain approximately the same. As described above, the utility function for the price is linear which is the reason for the linear course of the first part of the curve. At $U_{provider} \approx 72000$ the curve shape changes. In figure 7, until $U_{provider} \approx 72000$ the price has simply increased while all the other parameters remain almost unchanged. At $U_{provider} \approx 72000$ the price has been increased to the consumers limit. Thus, to further increase the utility for the provider all the other VM characteristics have to be changed. So RAM, storage and processing power are interchanged and reduced to get a provider utility which is greater than ≈ 72000.

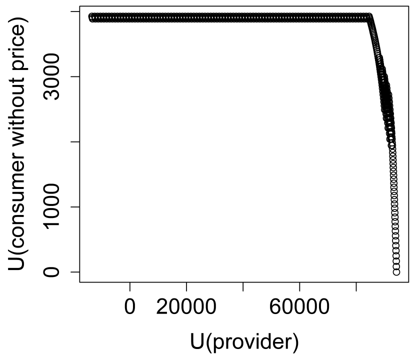


Fig. 8: Consumer utility without price

TABLE IV: Selected Pareto optimal points (approximation)

U(consumer)	U(provider)	VM characteristic
8824	3425	(995,58000,14.5,20)
8749	*4425*	(995,58000,14.5,21)
8724	5425	(995,58000,14.5,22)
7530	30775	(985,59000,15.75,48)
7531	30125	(990,59000,15,47)
7532	30050	(980,59000,15.25,47)

We also tried to capture the influence of the consumer Δ on the result. Therefore we run the scenario with a Δ of 200, 150, 100, 80, 60, 40, 30, 25, 20, 15, 10 and 5. The results are depicted in figure 9. The black line represents the consumers

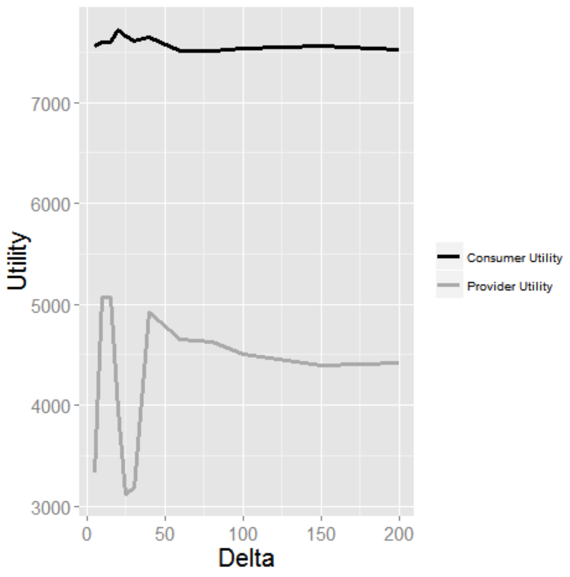


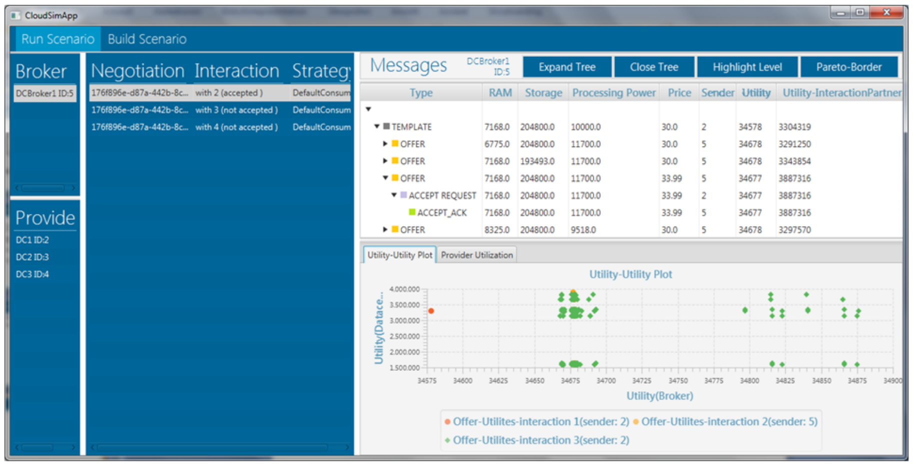
Fig. 9: Utility of agreement depending on Δ

utility value while the gray line represents the providers utility value. The Δ represents a similarity coefficient. A lower Δ value leads to counteroffers which are similar to received offer. A high Δ value leads to counteroffers which are more different to received offers. Very high deltas are not depicted as they often lead to no result.

The provider seems to be more affected by modifying the consumer Δ than the consumer. The figure shows that the utility is more unstable for a small Δ. This is because a small Δ usually leads to small utility changes of counteroffers received by the provider bringing the offers constantly closer to the provider acceptance utility of 3000. Sometimes the consumer creates counteroffers which are barley exceeding the providers acceptance utility . If no counteroffer is exceeding the provider acceptance utility then no agreement is formed. In such cases the provider creates counteroffers significantly improving its utility and sends them to the consumer. The instability results from offers barley exceeding a provider acceptance utility due to the usage of a small Δ. The usage of a small consumer as well as provider Δ would lead to more constant line.

VII. Implementation

Using our implementation we can build any scenario and adding new strategies. Figure 10 depicts a screenshot of our Bazaar-Extension. The screen illustrates the result of a Bazaar simulation scenario in CloudSim. The left side of the screen shows the providers and brokers (consumers) attending at the scenario. By clicking on a broker or a provider its negotiations and the used strategies are shown. Further, a utility-utility plot is created illustrating the utilities of offers for consumers and providers. The negotiation tree is illustrated as hierarchical list. The GUI is able to add the Pareto-Boarder and highlight the counteroffers of a user selected offer. We are currently expanding the Bazaar-Extension as described in the following section.



Negotiation	Interaction	Strateg
176f896e-d87a-442b-8c...	with 2 (accepted)	DefaultConsum
176f896e-d87a-442b-8c...	with 3 (not accepted)	DefaultConsum
176f896e-d87a-442b-8c...	with 4 (not accepted)	DefaultConsum

Type	RAM	Storage	Processing Power	Price	Sender	Utility	Utility-InteractionPartner
TEMPLATE	7168.0	204800.0	10000.0	30.0	2	34578	3304319
OFFER	6775.0	204800.0	11700.0	30.0	5	34678	3291250
OFFER	7168.0	193493.0	11700.0	30.0	5	34678	3343854
OFFER	7168.0	204800.0	11700.0	33.99	5	34677	3887316
ACCEPT REQUEST	7168.0	204800.0	11700.0	33.99	2	34677	3887316
ACCEPT_ACK	7168.0	204800.0	11700.0	33.99	5	34677	3887316
OFFER	8325.0	204800.0	9518.0	30.0	5	34678	3297570

Fig. 10: Result view of CloudSim Bazaar-Extension - Screenshot

VIII. Conclusion and Further Research

If we are assuming that in a supermarket based market mechanism the offer is equivalent to the template, the chance that this offer is Pareto efficient is small.We showed in the example that the negotiation result is better than the template in terms of utility. This is a general effect of negotiation: the negotiation gives both, consumer as well as provider the chance to improve offers. A general comparison between negotiation, supermarket and auction is difficult. Each consumer has different utility functions and each provider has different data centers and cost functions. Only simulation based approach can be used to make comparisons.

In our current counteroffer generator we produce a lot of offers which may be redundant or inappropriate. In our further research we focus on the reduction of messages exchanged. Moreover, we want to utilize the history database as well as model of the negotiation partner to further improve negotiation. The modelling of the negotiation partner seems to be challenging. We think that genetic programming approaches as well as neural networks may be appropriate to assess the VM valuations of offers. This network gets the characteristics of the virtual machines as input and responds with a assessed utility value. The presented simulation scenario can be extended too. We want to assess the behaviour in a n:m negotiation.

References

[1] J. F. Rayport and J. J. Sviokla, “Exploiting the virtual value chain,” vol. 73, no. 6, p. 75. [Online]. Available: ftp://218.31.79.211/%B5%E7%D7%D3%CD%BC%CA%E9/001/POIUYTREWQ403/F-%BE%AD%BC%C3/40741973069056505.pdf

[2] I. U. Haq, E. Schikuta, I. Brandic, A. Paschke, and H. Boley, “Sla validation of service value chains,” in *Grid and Cloud Computing, International Conference on*. IEEE, pp. 308–313. [Online]. Available: http://www.computer.org/csdl/proceedings/gcc/2010/4313/00/4313a308.pdf

[3] W. Mach and E. Schikuta, “A generic negotiation and re-negotiation framework for consumer-provider contracting of web services,” in *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*. ACM, pp. 348–351. [Online]. Available: http://dl.acm.org/citation.cfm?id=2428800

[4] P. Bonacquisto, G. D. Modica, G. Petralia, and O. Tomarchio, “A strategy to optimize resource allocation in auction-based cloud markets,” in *Services Computing (SCC), 2014 IEEE International Conference on*. IEEE, pp. 339–346. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6930552

[5] P. Samimi, Y. Teimouri, and M. Mukhtar, “A combinatorial double auction resource allocation model in cloud computing.” [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025514001054

[6] J. Chen, X. Han, and G. Jiang, “A negotiation model based on multi-agent system under cloud computing,” in *ICCGI 2014 : The Ninth International Multi-Conference on Computing in the Global Information Technology*. ICCGI, 2014.

[7] W. Mach and E. Schikuta, “Toward an economic and energy-aware cloud cost model,” vol. 25, no. 18, pp. 2471–2487. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/cpe.3086/full

[8] G. Whe and U. Dring, *Einfhrung in die Allgemeine Betriebswirtschaftslehre*. Vahlen.

[9] S. Zaman and D. Grosu, “Combinatorial auction-based mechanisms for VM provisioning and allocation in clouds,” in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. IEEE, pp. 729–734. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6217502

[10] O. Waeldrich, D. Battr, F. Brazier, K. Clark, M. Oey, A. Papaspyrou, P. Wieder, and W. Ziegler, “Ws-agreement negotiation version 1.0,” in *Open Grid Forum.[Accessed 20 May 2013]. Available from: http://www. gridforum. org/Public_Comment_Docs/Documents/2011-03/WS-Agreement-Negotiation+ v1. 0. pdf*.

[11] Amazon. Amazon EC2-spot-instances. [Online]. Available: http://aws.amazon.com/de/ec2/purchasing-options/spot-instances/

[12] A. F. M. Hani, I. V. Paputungan, and M. F. Hassan, “Renegotiation in service level agreement management for a cloud-based system,” vol. 47, no. 3, p. 51. [Online]. Available: http://dl.acm.org/citation.cfm?id=2716319

[13] N. Mankiw, *Principles of Economics*. Cengage Learning.

[14] J. Wilkes, “Utility functions, prices, and negotiation,” pp. 67–88. [Online]. Available: http://www.hpl.hp.com/techreports/2008/HPL-2008-81.pdf

[15] R. Buyya, R. Ranjan, and R. N. Calheiros, “Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities,” in *High Performance Computing & Simulation, 2009. HPCS’09. International Conference on*. IEEE, pp. 1–11. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5192685

Appendix

We implemented our negotiation based resource allocation mechanism in CloudSim. This appendix gives an introduction to CloudSim.

The CloudSim framework introduced in [15] was developed to model clouds. Based on these models, several allocation mechanisms can be simulated. Figure 12 shows some of the basic components used in CloudSim.

In CloudSim you can model so called data centers. A data center consists of one or more hosts whereby each host has several resources such as processing power, storage and RAM. We will consider these three characteristics as well as the price. In our paper, one data center represents one cloud provider. Virtual machines run on hosts and are characterised by the same properties as the hosts: processing power, storage and RAM. A host can run several virtual machines such as illustrated in figure 11. In this figure a host runs two virtual machines and allocates it's physical resources on them. A host can't allocate more resources to virtual machines as physical resources are available.

A virtual machine is owned by a consumer. The consumer is represented by a broker which is responsible for finding adequate virtual machines.

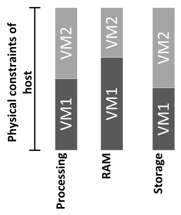


Fig. 11: Two virtual machines on a host

A. Allocation Mechanism

CloudSim offers several allocation policies which can be extended or replaced. Some of these allocation mechanisms are explained below: The numbers in figure 12 correspond to the numbers in the following enumeration.

1) The VM Allocation Policiy is responsible for assigning virtual machines to hosts. The default policy is the First-Come-First-Service policy [15]. So if a data center has to run a new virtual machine, one of it's host after the other is evaluated until a host is found which has the physical capabilities to run the virtual machine. In all our following examples we will use this policy.
2) A virtual machine can specify the processing power as well as the number of processors. The VM provisioning policy is responsible for the mapping physical processors of the host to the processors in virtual machines. CloudSim offers two different policies [15].
   The space shared policy assigns each physical processor to one virtual processor. Using this policy, a host with two processors isn't able to run a virtual machine with three or more virtual processors.
   Contrary to the space shared policy, CloudSim offers the time shared policy. Using this policy, a virtual machine is assigned a time slice of a processor. So one physical processor can be mapped to several virtual processors. Of course, they have to share the capacity.
3) The third allocation policy represents the focus of our research: the assignment of virtual machines to consumers. For this allocation, several economical aspects have to be considered. Interesting questions regarding this policy are: (i) *Q1*: Who gets the virtual machine if several consumers want to buy it? (ii) *Q2*: How is the price determined?

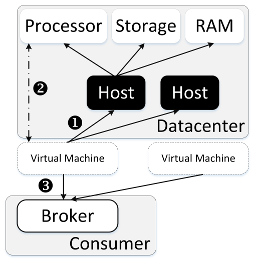


Fig. 12: Simplified CloudSim structure

B. Simulation Flow

CloudSim's simulation flow is based on events [15]. Events can be considered as messages which have a well defined structure. Basically, an event consists of a source, destination, tag and data. Of course there are more fields such as delay. In course of this paper, these fields are neglected.

The source represents the sending entity whereas the destination represents the receiving entity. Entities can for example be the consumers's broker or a data centers. It is also possible that entities send events to themselves. The tag can be interpreted as the message type which indicates what action the receiving entity should do. The data field can contain any data which the receiving entity may use.

The simulation flow is organized in phases. Before simulation starts, each entity can submit events to the simulation. After all events have been collected, the events are processed. By processing a event, new events can be send. Figure 13 shows such a scenario. The events which should be processed are stored in a so called deffered queue. By processing *Event A* three new events are create: *Event A*, *Event B* and *Event C*. Theses events are temporary stored in the future queue. After all events from the deffered queue have been processed, the events from the future queue are copied to the deffered queue and processed. If both queues are empty, the simulation is finished. For a detailed description of the simulation flow, read [15].

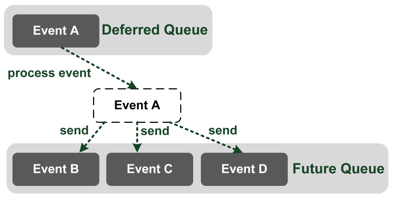


Fig. 13: CloudSim simulation flow