FairCloud - Auction-driven Cloud Scheduling

Artur José Lourenço Fonseca

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Luís Manuel Antunes Veiga
Prof. José Manuel de Campos Lages Garcia Simão

Examination Committee

Chairperson: Prof. Ana Teresa Correia de Freitas
Supervisor: Prof. Luís Manuel Antunes Veiga
Members of the Committee: Prof. David Manuel Martins de Matos

November 2017

Acknowledgments

I would like to thank my family for their friendship, encouragement and provide me the best conditions to fully focus in this thesis.

It is important to remind my Bachelor and Master of Science professors that taught me how to research and seek continuous improvement, and how to grasp knowledge from multiple subjects to achieve an ultimate goal.

I would also like to acknowledge my dissertation supervisors Prof. Luís Veiga and Prof. José Simão for their valuable insight and sharing of knowledge that has made this work possible.

Last but not least, to all my friends and colleagues that gave me multiple improvement suggestions and with whom I could discuss various topics.

To each one of you — Thank you.

Abstract

With Cloud Computing, access to computational resources has become increasingly facilitated, and applications can offer improved scalability and availability. However, those properties are supported by an energy consumption that is not proportional to the workload, and a pricing model that cannot be tailored to different types of users. One way of improving energy efficiency is by reducing the idle time of resources - resources are active but serve a limited useful business purpose. We present FairCloud, a Cloud-auction scheduler that facilitates the allocation by allowing the adaptation of Virtual Machines (VM) requests (through conversion to other VM types and resource degradation), depending on the user profile. Additionally, this system implements an internal reputation system, to detect providers with low Quality of Service (QoS). FairCloud was implemented using CloudSim and the extension CloudAuctions. It was tested with the Google Cluster Data. We observed that we achieve faster allocations and increased CPU utilisation, translating in more requests satisfied, improved energy efficiency and a higher provider profit. Our reputation mechanism proved to be effective by lowering the ranking of the providers with lower quality.

Keywords

Cloud computing; Energy; Pricing; Auctions; Scheduling; Quality of Service;

Resumo

Com a Computação em Nuvem, o acesso a recursos computacionais tornou-se cada vez mais simples, e as aplicações podem oferecer melhor escalabilidade e disponibilidade. No entanto, essas propriedades são suportadas por um consumo de energia que não é proporcional à carga computacional, e um modelo de preços que não pode ser adaptado a diferentes tipos de utilizadores. Uma maneira de melhorar a eficiência energética é reduzir o tempo estacionário dos recursos - os recursos estão ativos, mas atendem a um propósito comercial limitado. Apresentamos o FairCloud, um escalonador que facilita a alocação, permitindo a adaptação de pedidos de máquinas virtuais (através da conversão para outros tipos de máquinas e degradações), dependendo do perfil do utilizador. Além disso, este sistema implementa um sistema de reputação interna, para detetar fornecedores com baixa qualidade de serviço. O FairCloud foi implementado usando o CloudSim e a extensão CloudAuctions. Foi testado com o Google Cluster Data. Observámos que conseguimos alocações mais rápidas e aumento da utilização de CPU, traduzindo-se em mais pedidos satisfeitos, mais eficiência energética e um maior lucro do fornecedor. O nosso mecanismo de reputação mostrou-se eficaz reduzindo a classificação dos fornecedores com menor qualidade.

Palavras Chave

Computação em nuvem; Energia; Modelos de preço; Leilões; Escalonamento; Qualidade de serviço;

Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface
AWS	Amazon Web Services
CMS	Cloud Management System
COCA	Compatible Online Cloud Auction
CPU	Central Processing Unit
EC2	Elastic Compute Cloud
IT	Information Technology
MB	Megabyte
MIPS	Million of Instructions per Second
PTN	Price and Time-slot Negotiation
QoS	Quality of Service
RAM	Random Access Memory
SaaS	Software as a Service
SLA	Service Level Agreement
UML	Unified Modelling Language
VM	Virtual Machine

1 Introduction

Contents

Cloud Computing is defined as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [7]. However, those properties are supported by an energy consumption that is not proportional to the workload and a pricing model that cannot be tailored to different types of users (e.g. job-oriented, resource-aggressive, relaxed).

Cloud Computing revolutionized the way Information Technology (IT) resources are managed [8]. First, it introduced the concept of infinite computing resources available on demand (on the user perspective), which made it possible to handle unexpected loads. Also, there was no up-front commitment. Cloud computing benefited developers and small companies that can rent only the services needed and increase them as their business grows. Finally, cloud computing uses a payment model with low granularity (e.g. seconds, minutes, hours) and users can release resources when they are no longer useful.

The major component of the Cloud Computing is the datacenter. Approximately 25% of the total corporate IT budget is spent in datacenters, including management and facilities [9]. A datacenter has four fundamental components:

- Network — datacenter, inter-datacenter network, end-user network;
- Servers — enclosure, racks, components;
- Cloud Management System — scheduler, monitoring system, virtualisation;
- Appliances — application, runtime environment, operation system.

In this work, we will focus on studying and improving a Cloud Management System (CMS). It is composed of a monitoring system, a virtualisation technology and a scheduler. A monitoring system is needed to provide information about current allocations and available resources. Virtualisation technology offers more customisation: better resource management and increased software portability. The scheduler's primary function is to deploy the resources requested by the user.

In particular, a scheduler needs to consider the priority of customers, job length, consumed energy, job revenue, cancellation and migration costs, and the low-level information provided by the monitoring system.

The following sections describe two issues that will be discussed in this thesis: energy consumption and cloud pricing models, and that comprise the challenges we aim to address with our work.

1.1 Challenges

The challenges are the energy and the cloud pricing.

1.1.1 Energy

Datacenters have a high energy consumption. In this work, we will study the energy consumption and analyse ways of increasing their useful output. Figure 1.1 gives us details on how the energy is consumed. We observe that half of the energy is used in cooling, followed by server and storage, and power conversion.

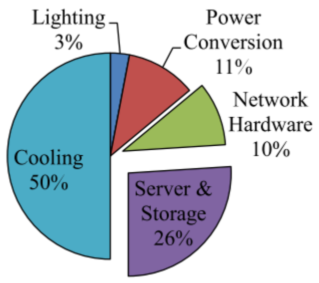


Figure 1.1: A breakdown of energy consumption by different components of a datacenter [1].

As detailed in the work in [10], energy efficiency can be achieved in multiple ways:

- Implement efficient components;
- Reduce overhead of supporting systems (e.g. cooling cost);
- Reduce idle run of the system, increasing utilisation (i.e. increased energy consumption but producing relatively more output);
- Avoid redundant/unnecessary operations.

1.1.1.A Sub-utilisation

Studies suggest that 30% of the servers are on idle, with less than three percent average daily utilisation [9]. They still consume energy but serve a limited useful business purpose. Within a datacenter, the typical average daily server utilisation is 6%, creating a capital loss. In Figure 1.2, we see during the majority of the time, servers operate between 3 and 50% Central Processing Unit (CPU) utilisation. They are rarely completely idle, but almost never at maximum utilisation.

Figure 1.3 shows the number of active workstations in use at RNL, Instituto Superior Técnico, Portugal. It represents a smaller cluster of 96 machines. Similarly, we observe sub-utilisation - the average utilisation is 23 machines.

Previous studies [11] and [2] address the reasons for systems being in idle. Even during periods of low service demand, servers cannot be terminated because they still run background tasks. Computational tasks can be transferred but distributed databases and file systems are difficult to transfer. Those

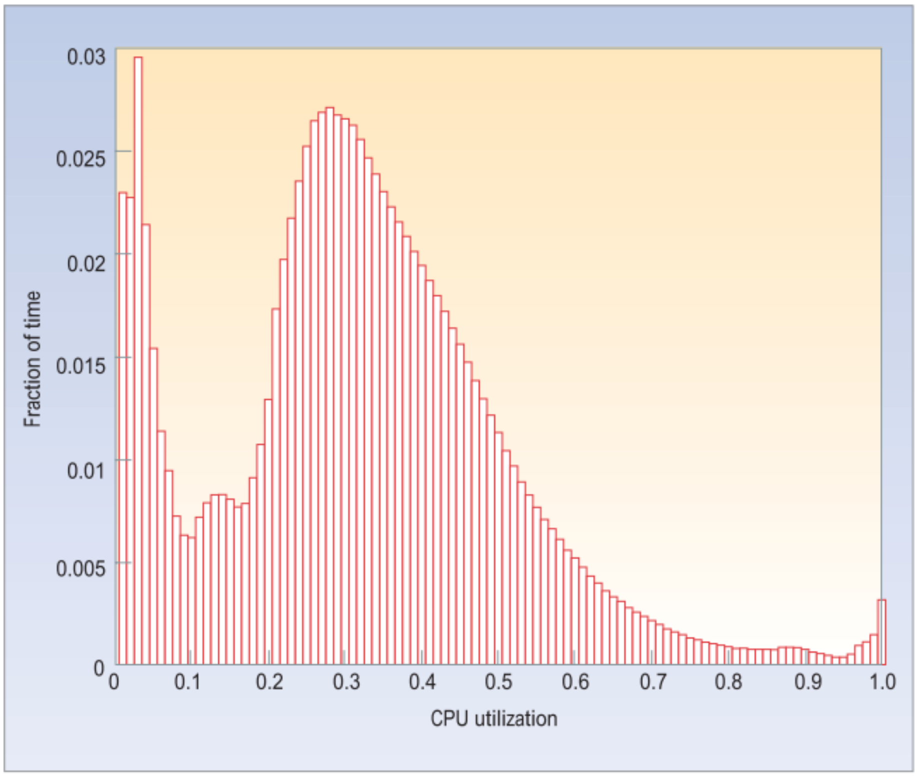


Figure 1.2: Average CPU utilisation of more than 5,000 servers during a six-month period [2].

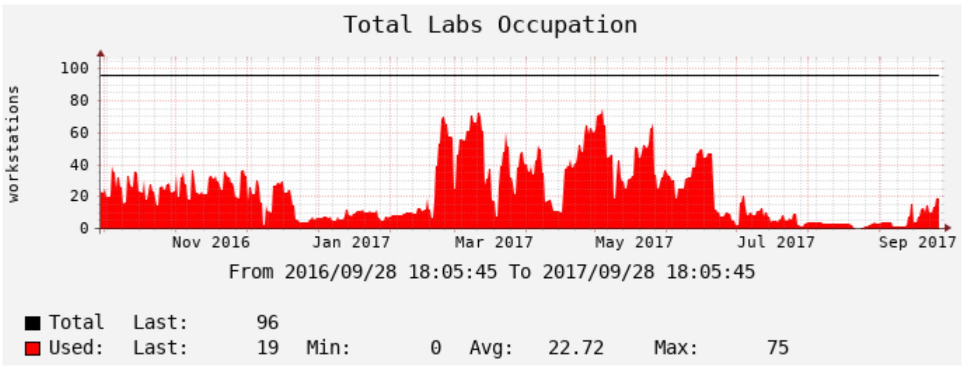


Figure 1.3: Workstations occupation over a period of one year, at RNL [3].

tasks are fundamental to improve fault tolerance (not offered by a central server) and to increase the speed of recovery after a crash. One example of these file systems is Google File System [12]. Also, servers perform small network tasks.

On the other hand, applications cannot run on fully utilised servers, as even non-significant workload fluctuation or any internal disruptions, such as hardware or software faults, will lead to performance degradation and failing to provide the expected Quality of Service (QoS).

Energy efficiency is the ratio of *useful output* per *amount of resources consumed*.

Energy-proportional computing is a model where *Utilisation* and *Server power usage* are directly related [2]. The idle reduction is based on this model. Furthermore, the idle optimisation can be made at different levels: CPU, cluster and between datacenter. In Figure 1.4, the green line represents the server power usage, in a typical server, at varying utilisation levels. We observe that it is consuming already half of the power with very low utilisation. The energy efficiency (red line) increases as the utilisation increases. The system represented in the graphics is not energy-proportional.

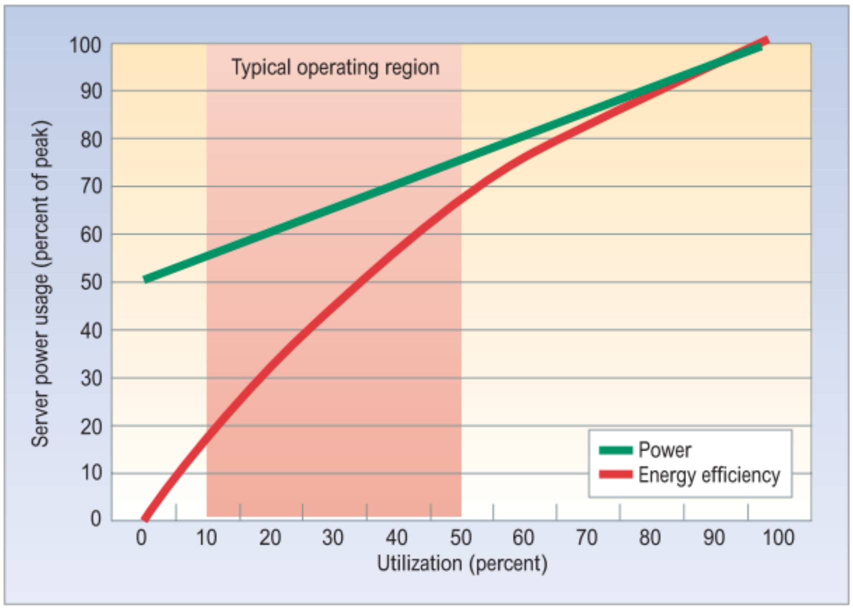


Figure 1.4: Server power usage and energy efficiency at varying utilisation levels [2].

1.1.1.B Two Examples of Optimisation at Different Levels

To explain the concepts of energy optimisation at CPU and datacenter level, we provide two examples.

Optimisation in the CPU The Workload Dependent Dynamic Power Management proposed in [13] improves multi-core server performance and reduces energy consumption. The system is based on a queue model that minimizes the average task response time. This work formulates two speed models. In idle-speed, a core runs at zero speed when there is no task to perform. In the constant-speed model, all cores run at the same speed even if there is no task to perform. The system demonstrates that for the same average power consumption, it is possible to design a multi-core processor so that the average task response time is shorter than a multi-core processor with a uniform clock rate. In other words, it is possible to schedule tasks in a way that average task response time is reduced, maintaining the same average power consumption.

This Dynamic Power Manager contributes to the energy-proportional computing because background tasks can be done in one specific core while others are in sleep mode, maintaining a similar overall performance.

Optimisation between datacenters The system presented in [14] studies the impact of geo-temporal inputs in energy consumption. Those inputs are electricity price, the availability of renewable energy resources, and the cooling efforts (e.g. low temperature, wind speed). Ideally, a Virtual Machine (VM) should run in the location with the cheapest energy and the lowest temperature. However, the transfer overhead must be considered because it might affect QoS and availability.

Furthermore, this contribution formulates the constraints based on the inputs and tries to optimise them in two stages: global and local optimisation.

1.1.2 Cloud Pricing

Cloud providers can apply different pricing models, as described in [15]. In the **Pay-as-you-go** model, often referred to On-Demand, the user is charged hourly without any long-term commitments or upfront payments. In the **Subscription** model, the total price is reduced, but an upfront payment and a long-time commitment (e.g. one year) are required. This approach might be beneficial for the user or the provider, depending on resource usage. Another commonly implemented model is **Pay-for-resources**. The price is based on resource consumption (e.g. DiskReadOps, NetworkIn). Even though it is a fair system, the monitoring system is hard to implement, and the user cannot be sure that the measurements are correct.

These models follow a fixed price approach that raises many problems. It is not possible to reflect demand and supply trends (i.e., not market-driven). It does not benefit from users that are willing to pay more. Furthermore, overpricing can lead to resource waste.

Recently, Google Cloud Platform and Microsoft Azure developed a pricing model for VMs without a minimum QoS guaranteed, but at a lower fixed price. Amazon Web Services (AWS) offers spot instances where the prices dynamically change depending on the supply-demand conditions. Table 1.1 allows us to compare the different pricing models by comparing the AWS Elastic Compute Cloud (EC2) pricing models. These prices are available online and were recorded on 16/09/2017 for the Europe (Ireland) location. They represent three categories of instances: general purpose, compute optimised, and memory optimised, respectively.

Table 1.1: Hourly prices for different VM types, in AWS.

Type	On-Demand	Subscription (1-year)	Spot
m4.large	$0.111	$ 0.071	$0.0191
c4.large	$0.113	$0.073	$0.0262
r3.large	$0.185	$0.114	$0.0214

Despite the improvements regarding traditional systems, the dynamic pricing models implemented by Google, Microsoft, and Amazon have multiple shortcomings:

- Still difficult to adjust the price to supply and demand conditions;
- The price does not reflect the QoS directly (i.e., it does not depend on the real offered quality);
- Only two types of users: ”regular” (receive all resources) or ”low priority” (e.g. the execution may terminate abruptly);
- Only suitable for applications with asynchronous tasks and with fault-tolerance;
- No guarantees when the allocation terminates.

1.2 Contributions and Objectives

We create a Cloud-auctions mechanism that works like a CMS between datacenters. By lowering the cost paid per VM, the datacenters will receive more load, increasing their utilisation, and hence contribute to more efficient energy usage.

A dynamic pricing algorithm must be used to accommodate heterogeneous users (e.g. low budget, strict time restrictions), providers (e.g. domestic networks, universities, research labs, enterprise) and market conditions (i.e., supply and demand). As a side effect, the allocation and execution time will be reduced because the load/demand will be degraded — assuming the user profile allows it.

The proposed objectives are:

- Analyse previous research works in Cloud-auctions and related topics (e.g. auction theory, scheduling, energy efficiency).
- Formulate a taxonomy and classify the systems according to it;
- Propose an algorithm respecting the desired properties (i.e., improve energy efficiency, reduce allocation time, and use a dynamic pricing);

- Implement a prototype following the algorithms;
- Prepare datasets with data obtained from real requests to Google Cluster Data;
- Evaluate the prototype and present the results.

The work presented in this thesis is partially described in ”A. Fonseca, J. Simão, and L. Veiga, “Faircloud: Truthful cloud scheduling with continuous and combinatorial auctions,” in Cloud and Trusted Computing — International Symposium on Secure Virtual Infrastructures. Springer LNCS, 2017”.

1.3 Document Structure

This thesis is organised as follows: Chapter 2 presents and analyses the related work. Chapter 3 describes FairCloud, including the main entities and their interaction, algorithms, and implementation. Chapter 4 explains the evaluation methodology and presents the results. Finally, Chapter 5 presents concluding remarks.

2 Related Work

Contents

In this chapter, we describe the cloud-auctions taxonomy, the scheduling taxonomy, and present relevant research and commercial systems.

2.1 Cloud-Auctions Taxonomy

We constructed this taxonomy by analysing the literature and the differences between the systems. Any system, including the one we will develop, can be described based on this taxonomy.

The main participants in a cloud-auction are the user and the provider. They are interested in buying and selling computational resources, respectively.

Cloud-auctions algorithms have different categories. The full taxonomy diagram is displayed in Figure 2.1.

2.1.1 Resource Type

The first aspect to consider when designing a cloud-auction is what will be traded. The systems can be tailored for a particular resource type or generic. The majority of systems, [4, 16–19] and AWS EC2, focus on VM, including its different kinds: general purpose, compute, memory, and storage optimised, and accelerated computing (e.g. machine learning, 3D rendering). Ginseng [20] trades physical memory. Finally, OptiSpot [5] considers applications as a whole and not their resources individually.

2.1.2 Negotiation

Regarding the negotiation, auctions can be classified according to the participants and the bidding scheme.

2.1.2.A Participants

Single In single auctions, only one participant (either user or provider) submits bids. There are traditionally four types of single auctions that are commonly used in non-automated contexts [21]. In the English auction (or open ascending price), the users bid openly against each other. Each bid must be higher than the previous one, and the provider may set a minimum interval between bids. The highest bidder pays their bid if it is higher than the reserve price — minimum price set by the provider. It is used in artwork and real estate. In the Dutch auction (or open descending price), the provider lowers the price until a user is willing to accept it. If the user does not bid the total amount, the auction will continue with the remaining amount. Not widely used, except for flowers and fish.

On the other hand, on sealed first-price auctions, the bids are private, and users cannot adjust their bids. The highest bidder pays the price they submitted. A typical use is in government contracts. Vickrey

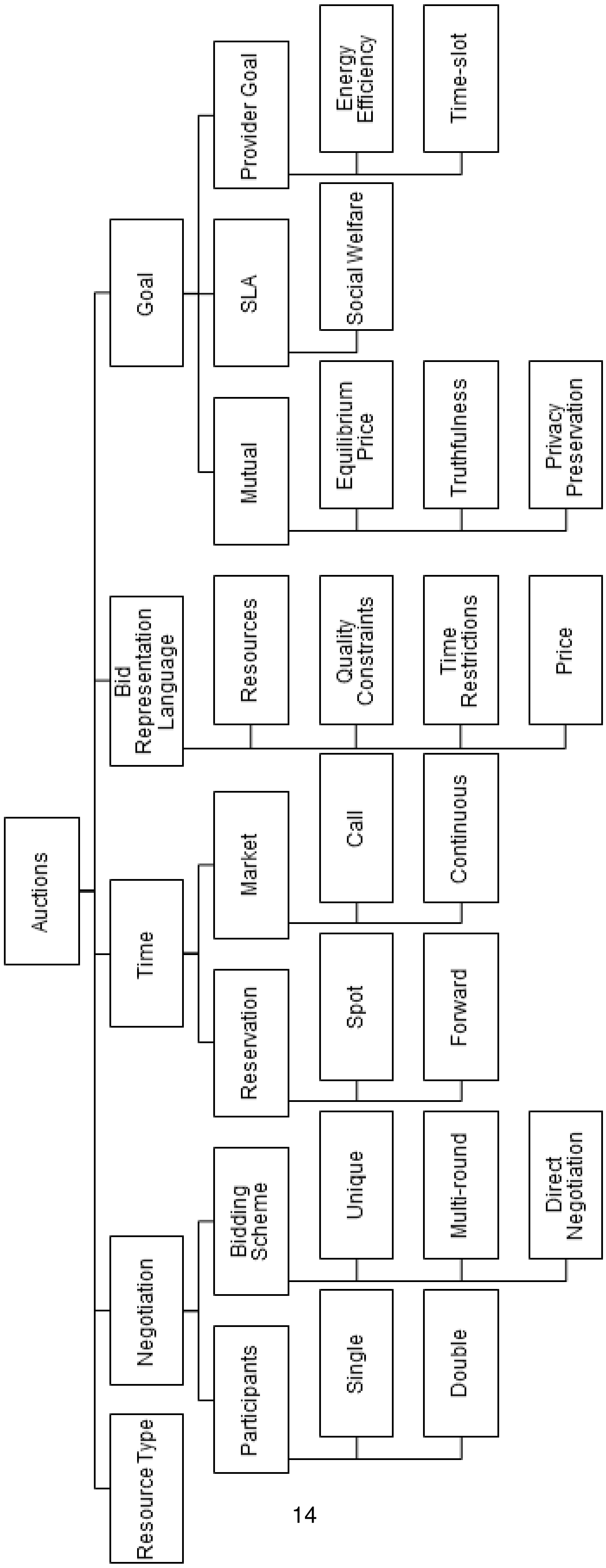


Figure 2.1: Cloud Auctions taxonomy.

auctions (or sealed second-price) operate similarly. However, the winning user pays the second highest price. Only used in automated contexts such as bidding for online advertising.

The commercial system AWS EC2 Spot Instances [22] implements a single sealed auction - only users bid. The research systems [6, 20] implement a single auction but not one of the four types.

Double In double auctions, both participants bid until an agreement is reached. The main difficulty is to find a competitive equilibrium - a price in which the supply equals the demand. This type of auctions is conducted by an independent participant - the auctioneer. It operates the auction by accepting and controlling the bids.

The majority of the research systems, namely [4, 16, 17, 23], use a double strategy.

2.1.2.B Bidding Scheme

Bidding scheme defines the process of submitting bids, and the possibility of improving a bid if it was not successful.

In the systems with **unique** bidding [4, 5, 16], each bid is only submitted once, but stays in the system and can be allocated later unless a deadline is specified.

A **multi-round** scheme is implemented in [17, 20, 23] but with small variations. In the first, the negotiation only terminates when the participants stop updating their bids, and each bid represents a new round. The second is similar, but the updates are only accepted if its price exceeds the current prices. Finally, on Ginseng the bids are deleted in each round if they are not allocated.

A Price and Time-slot Negotiation (PTN) system, proposed in [6], implements a **direct negotiation** scheme. Each participant registers and consults a centralised registry to obtain the communication details. Furthermore, the participants must implement protocols to generate and evaluate the bids.

The downside of the multi-round and direct negotiation schemes is the participants need to be active after submitting the initial bids if they wish to improve their bid.

2.1.3 Time

Regarding the time dimension, auctions can be classified according to the reservation policy and the market type.

2.1.3.A Reservation

The reservation can be in **spot** or **forward**. In spot auctions, the allocation starts immediately, unlike forward auctions. A typical user of the forward auction is the bank, which performs daily computations during two hours at midnight. This way, they benefit from renting that time slot in advance.

The algorithms [5, 16, 17, 20, 23] are spot auctions. In [6, 24] the user can bid for a future spot.

In other systems, like Compatible Online Cloud Auction (COCA) [4], the auctioneer is free to decide when the allocation will take place, but considering the user restrictions (available time and deadline).

2.1.3.B Market

The market is denominated **call** if it waits for a set of bids and processes them as a batch. Usually, it occurs in regular periods of time. The call interval can vary from minutes to days. It depends on the bid incoming speed and the participant tolerance to wait. Many stock exchanges operate a call market for trading in less active stocks or at opening (e.g. all orders placed during the night are processed at 8 am). Ginseng [20] is a call auction. AWS EC2 Spot Instances is also a call market but with shorter periods (e.g. 5 minutes).

In **continuous** auctions, also called online, the bid is processed as soon as it arrives. The examples are [4–6, 16, 18, 23].

Particularly, [24] is simultaneously a call and continuous market. The spot bids are handled continuously, but the forward bids are processed as a call once a day.

2.1.4 Bid Representation Language

A bid representation language is necessary to capture the participant's features. It should be concise and consistent. The four main points are resources, quality constraints, time restrictions, and price.

A comprehensive bid language benefits the participants: the bid information is sufficient to make a decision. They can express their needs and go offline, avoiding negotiations.

2.1.4.A Resources

The participant can choose the resource type and its amount. An auction is called combinatorial if the participants can bid in combinations of items (i.e. bundles). AWS EC2 and [5] only allow individual requests. A variety of systems allow the specification of a fixed amount. The systems [4, 20] allow bids with of a range of values.

2.1.4.B Quality Constraints

Recently, the auctioneers started to tailor their algorithms to user's quality constraints and the characteristics of the providers.

COCA, proposed in [4], implicitly considers three types of users. Depending on the type of user, the time restriction and price parameters in the bid language are different. A `(I) job-oriented` user has a fixed-size job and should be completed by a specific date. A `(II) resource-aggressive` user goal is to get a sufficient number of resources in a specific time period (e.g. Software as a Service (SaaS) during rush-hour). The last type is the `(III) resource-aggressive users with time-invariant requirements`, and they benefit in getting the maximum of resources within a preferred time duration.

The system [5] requires the user application workload and the desired maximum mean response time, and the provider's specification (number of resources and the request process rate). The bid representation language in [19] requires only the provider energy consumption per VM type. The system proposed in [17] addresses the problem of auctions on geo-distributed datacenters with different QoS. Therefore, the provider can specify QoS metrics (e.g. latency, uptime).

2.1.4.C Time Restrictions

In most systems, particularly in spot auctions, time restrictions cannot be specified [16–18, 20, 23].

However, in [4, 6, 19, 24], it is possible to specify the available time (i.e., when it is possible to start the job), deadline and job duration. The system [5] complements those systems by letting the provider specify a valid time for its resources.

2.1.4.D Price

Some systems allow the specification of a bid total price [6, 16, 18, 23, 24]. Other systems are more detailed and specify the cost per unit (or cost per unit per time unit) [17, 20] and AWS EC2.

The system [19] is motivated by the users with soft deadlines (i.e., non-critical). They can specify a price function based on elapsed time, usually with a constant price if the execution concludes before the deadline and a penalty for each time unit passed after the deadline.

COCA, proposed in [4], follows the same motivation but specifies parameters for the three users described in Section 2.1.4.B. Figure 2.2 represents the price function for each type. In (a), we observe that the price is constant until the deadline when a penalty is applied. In (b), the price depends only on the total allocated resources between the available time and the deadline. For the third user type, in (c), the function also depends on the allocated resources but multiplied by the time length. Summarising, the price parameters in the bid are:

- Type I: total price, penalty rate;
- Type II: price per number of resources in the valid interval;
- Type III: price per number of resources per time unit.

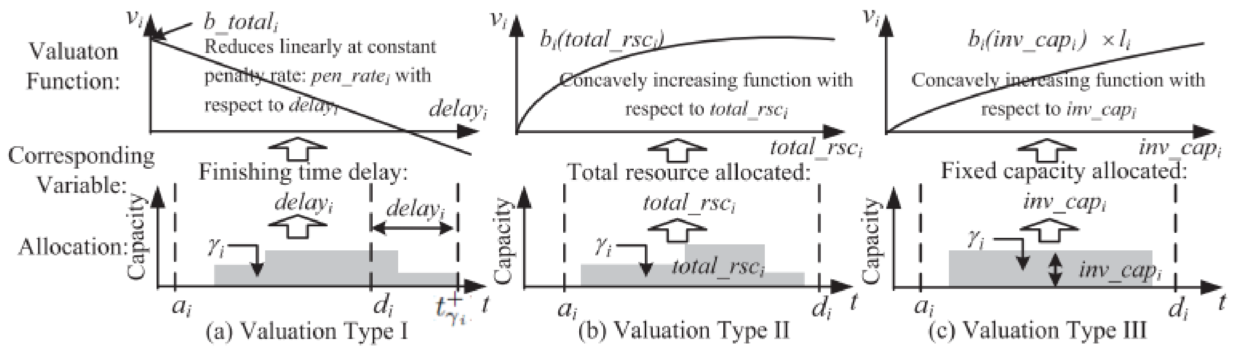

Valuaton Function:
Corresponding Variable:
Allocation:
v_i
b_total_i
Reduces linearly at constant penalty rate: pen_rate_i with respect to $delay_i$
$delay_i$
Finishing time delay:
$delay_i$
γ_i
a_i
d_i
$t_{\gamma_i}^+$
t
Capacity
(a) Valuation Type I
$b_i(total\_rsc_i)$
Concavely increasing function with respect to $total\_rsc_i$
$total\_rsc_i$
Total resource allocated:
(b) Valuation Type II
$b_i(inv\_cap_i) \times l_i$
Concavely increasing function with respect to inv_cap_i
inv_cap_i
Fixed capacity allocated:
(c) Valuation Type III

Figure 2.2: Price function for the different types of users [4].

2.1.5 Goals - Mutual

The auction algorithms try to find a compromise to satisfy the user and the provider. We will start by studying the properties desired by both the user and provider.

2.1.5.A Equilibrium Price

The equilibrium price calculation is important, and both participants are interested that their bid is reflected in the calculation.

The most straightforward approach is doing the average between user and provider price [16,17,23]. In PTN, the participants have total control on the final price as they have to accept it.

Ginseng [20] implements an exclusion compensation principle. In other words, it measures what would be the remaining users' allocated amount, if a particular user did not join the auction. Additionally, the user pays a fixed price for the base allocation.

OptiSpot [5] constructed a pricing resource model (with a bid value as the input parameter) and populated it using a third-party pricing history (AWS EC2 Spot), which contains the timestamp and spot prices for each resource type. The model contains the following data:

- Overbid time — time when a bid is the highest;
- Underbid time — time before a bid is the highest bid again;
- Average cost — cost to keep a resource active;
- Availability — percentage of time the resource is active.

The final price is the bid that optimises those factors. For example, a very high bid would maximise the overbid time and availability, but the average cost would be too high.

2.1.5.B Truthfulness

Truthfulness is the property of algorithms that incentive participants to reveal their true valuation. A typical example of a not truthful auction is the sealed first-price. Supposing that the user true valuation is v. By bidding higher than v, the result will be a loss. Bidding exactly v does not bring any benefit. However, by bidding below v, the user may have a positive gain (assuming it can win the auction).

Some systems formally proved that they are truthful [4, 17–19].

Finally, [5] and AWS EC2 Spot are not truthful, but they consider non-truthful bidding should also be allowed as the participant's strategy.

2.1.5.C Privacy Preservation

Data security and privacy in cloud computing are widely studied [25, 26] but still represent a major concern and it is mandatory in government data and the defence industry.

It is desired that cloud auctions consider the privacy preservation. The authors in [27] formulated the following threat model:

- A **bidder** may get to know others willingness to pay, and choose to bid untruthfully to get extra profit, thus tampering with the truthfulness;
- The **provider** can adapt its pricing strategy based on the bidders' bids to obtain extra profit;
- An **attacker** can impair the auction process by submitting a bid that cannot win the auction but will increase the price paid by the winners.

Ultimately, the only public information should be the winner ID, price, and requested bundles of VM instances. The Privacy Preservation framework described in [27] protects against those threats.

2.1.6 Goals - SLA

Focusing on the user desired properties.

2.1.6.A Social Welfare

Social welfare is not only related to having the QoS met but also the service quality and the auctioneer utilities and features.

In Ginseng [20], there is a guaranteed base (i.e., if the bid is not successful, a minimum memory is guaranteed) and, in the case of similar bids, the current memory holders are preferred to avoid the migration costs. In [17], the providers are rated and order by QoS. For example, users can be matched with providers geographically closer (i.e., less latency). However, the user cannot choose the location directly. It might prefer a location with more latency, but that assures redundancy.

Other systems support different categories of users. In PTN [6], the participants can reject the price and time slot before the allocation starts.

Some systems, like [23], implement strategies to converge the user and provider bids through belief functions (i.e., the chance of a bid getting accepted).

AWS EC2 and Google Cluster provide a price history for the user to perceive the typical trading prices, but the VM terminates (after a two-minute warning) if the spot price increases above the bid.

Finally, OptiSpot [5] decides what resources to rent and how to map them to the application modules. Figure 2.3 suggests three possible deployments and OptiSpot determines the best deployment and auctions those resources, following the user constraints (Section 2.1.4.B).

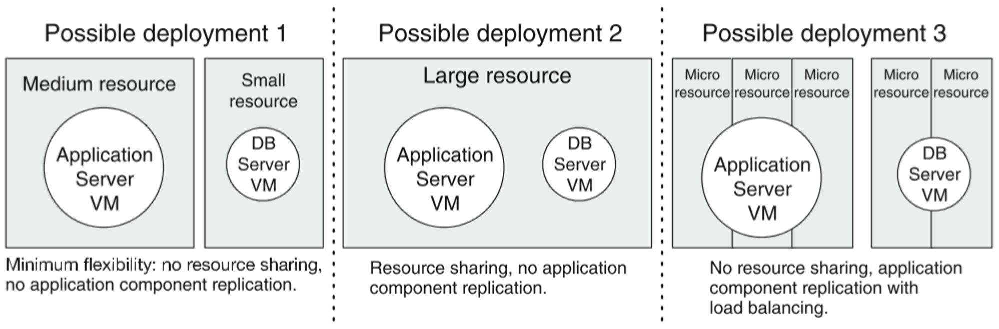


Figure 2.3: Different strategies for deploying the application on OptiSpot. Source: [5].

2.1.7 Goals - Provider

Finally, the provider is interested in energy efficiency and controlling the time-slots.

2.1.7.A Energy Efficiency

High energy efficiency is directly related with provider profit. The auction algorithm can perform the allocations in a way that the overall energy efficiency is maximised. However, just by using an auction the numbers of allocations increase, and consequently, the energy efficiency.

The system described in [19] considers the energy consumption and tries to find the optimal rate between operational costs and the number of allocations.

2.1.7.B Time-slot

Simultaneously, the provider is interested in the time-slot allocation, based on the following heuristics:

- When the service demand is lower, so the highly demanded times are reserved for high-paying users;
- As soon as possible because computing resources devalue with time;
- Best-fitting to optimise resource utilisation and, consequently, higher energy efficiency.

The majority of the systems allocate as soon as possible, considering the available time. The PTN system [6] negotiates the time, and the provider can follow one of the three heuristics. In [4], the chosen time-slot is the one that maximises the user's valuation.

2.2 Scheduling Assessment

Cloud-auctions systems can be assessed with typical scheduling criteria. The taxonomy proposed in the survey [28] is represented in Figure 2.4, and will be detailed in this section.

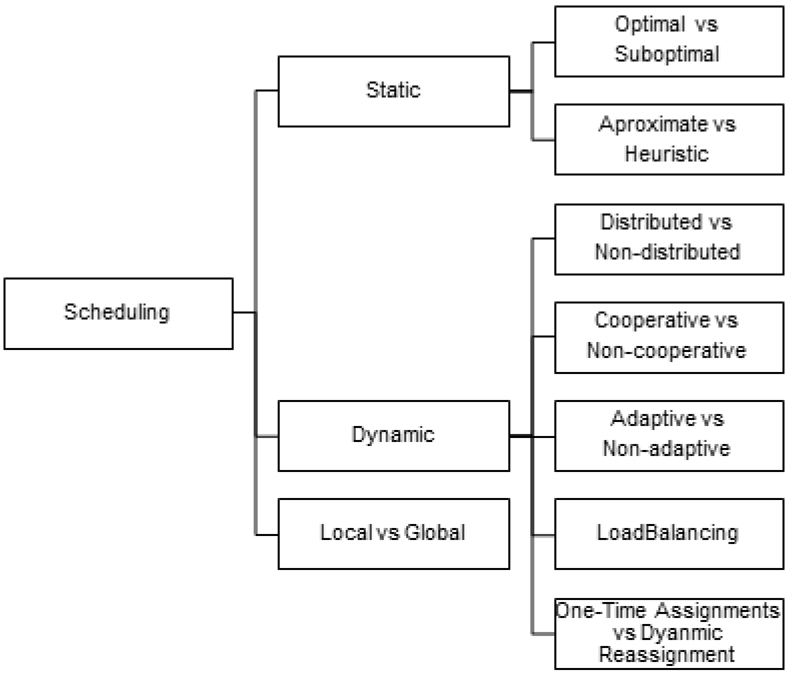


Figure 2.4: Scheduling taxonomy.

We observe that a scheduler can be assessed in two core categories: static vs dynamic, and local vs global. In static scheduling, all the information from the system is available at decision time. In dynamic scheduling, the resources needed by the process and the environment are unknown at the start.

Additionally, local scheduling is related to the assignment and sequencing of job in a single-processor. Global scheduling is the problem of deciding where to execute a job.

It is possible to further study the static and dynamic scheduling. In the end, we will evaluate a scheduler using this taxonomy.

2.2.1 Static Scheduling

Static scheduling is analysed in two dimensions.

2.2.1.A Optimal Versus Suboptimal

In Optimal scheduling, assignments are based on criteria functions (e.g. minimise energy consumption and process completion time). Those restrictions are, usually, computationally infeasible. Suboptimal approaches are not perfect but can be done in less time.

2.2.1.B Approximate Versus Heuristic

The first uses the same formal model as the Optimal solution, but the solution space is restricted. Heuristics are realistic assumptions known before system execution. Two typical heuristics are: groups of processes that communicate between them should be on the same processor and jobs that benefit from parallelism should be on different physical processors.

2.2.2 Dynamic Scheduling

Dynamic scheduling is more complex and will be studied in multiple dimensions.

2.2.2.A Distributed Versus Non-distributed

This decision is related with where the scheduling is made: among multiple processors or on a single processor. The first option raises the question of the role and authority of each scheduling process.

2.2.2.B Cooperative Versus Non-cooperative

In the cooperative case, each processor has the responsibility to carry out its portion of the scheduling task, but all processors are working toward a common system-wide goal. In the non-cooperative case, individual processors act alone as autonomous entities.

2.2.2.C Adaptive Versus Non-adaptive

An adaptive solution to the scheduling problem is one in which the algorithms change dynamically based on previous decisions and the resulting behaviour (e.g. algorithm ignores a parameter if it is not providing reliable information). In contrast, a non-adaptive scheduler does not modify the algorithm (e.g. all system inputs have always the same weight).

2.2.2.D Load Balancing

Ideally, the load on all processors should operate at the same rate. This task is facilitated on homogeneous nodes since they know the structure of the other nodes. For load-balancing to occur, it is necessary to decide the appropriate measures for load, and each node must provide them accurately.

2.2.2.E One-Time Assignment Versus Dynamic Reassignment

In One-time assignment, the scheduling decision is made once for each job. In contrast, dynamic reassignment tries to improve earlier decisions by rescheduling. Rescheduling is motivated by users not providing accurate information (e.g. a task is longer than the user said it would be) or the environment changed (e.g. new processors are available). Reassignment is only possible if the benefits overcome the migration price.

2.2.3 Partial Utility-driven Scheduling

The scheduler proposed in [29] is motivated by the current cloud provider's Service Level Agreement (SLA) limitations. It is all-or-nothing: either the user gets a fixed compensation, if the availability falls below a certain threshold, or does not receive anything. Furthermore, for a user to receive the compensation, it needs to fill a complaint, and it will come in the form of compensation credits to be used in the following charging cycles.

The unit cost is calculated by multiplying the VM price (defined by the provider), the degradation factor, and the partial utility value. The degradation factor is the real amount of resources allocated. This research introduced a bi-dimensional matrix that outputs the partial utility based on the user category (i.e. high, medium, low) and the level of resource degradation interval (e.g. [0%, 20%[).

This work contains a formula to obtain the Aggregated Degradation Index. It is an internal metric, used by the provider, to control the maximum overcommitment which, on average, is applied to the clients.

The system also presents the allocation algorithms, receiving a VM and a list of hosts as input. The first algorithm searches for the host with either more or less available cores, depending on the order criterion. It is called first-fit increasing/decreasing. If the base allocation is not possible, an overcommitment is necessary. The utility allocation algorithm orders the hosts by available resources (ascending). For each host, the algorithm calculates the needed amount and runs the selection algorithm that chooses which and how much the VMs will be degraded (from those already allocated). Finally, it applies the degradation to the VMs in the selection. Degradation only happens if the Aggregated Degradation Index does not exceed the maximum limit.

Detailing the select algorithm: it consists of a main loop that stops when all the requested power is achieved. Inside the loop, the machines are continuously degraded to the next category (e.g. 0% to 10%).

We can now characterise the Partial Utility-driven Scheduling:

- Global — The goal is to decide where to allocate the VM;
- Dynamic — Future information is not known (e.g. new requests may arrive);
- Distributed, Cooperative — The allocation is divided into multiple hosts, and all cooperate to allocate the maximum possible resources;
- Non-adaptive — The algorithm does not modify itself;
- Load Balancing — Not implemented, the allocation goes to the first-fit always;
- Dynamic Reassignment — A current allocation may be degraded but is not migrated to other hosts.

2.3 Relevant Research Systems

In this section, we detail some systems that combine multiple properties from the taxonomy and implement a more complex architecture. In particular, we mention direct negotiation, pricing formulas, bidding strategies, and privacy preservation.

2.3.1 PTN

The PTN system, described in [6], implements direct negotiation. The first step in generating and evaluating (price, time-slot) pairs is creating the utility function. The price and time utility functions represent the gain by dealing at that price or in that time-slot, respectively.

The time-slot utility is different for each participant:

- User — chooses a set of preferred time-slots (i.e., utility is maximum) and a monotonically increasing and decreasing period before/after it (i.e., neighbourhoods of the preferred time-slots). In this period, the utility is lower but still acceptable;
- Provider — attributes weights to three types of slots: low-demand, early jobs, and fitting job sizes.

After the functions are defined, the participants send concurrent pairs of time and price values. The other participant computes the total utility (by attributing weights to price and time and summing) and reaches an agreement if it exceeds a predefined threshold. Figure 2.5 (a) represents a proposal with seven different pairs but with equal total utility. Figure 2.5 (b) represents the evaluation of the proposal. We observe that the pair 5 is accepted because it represents the highest utility for the provider.

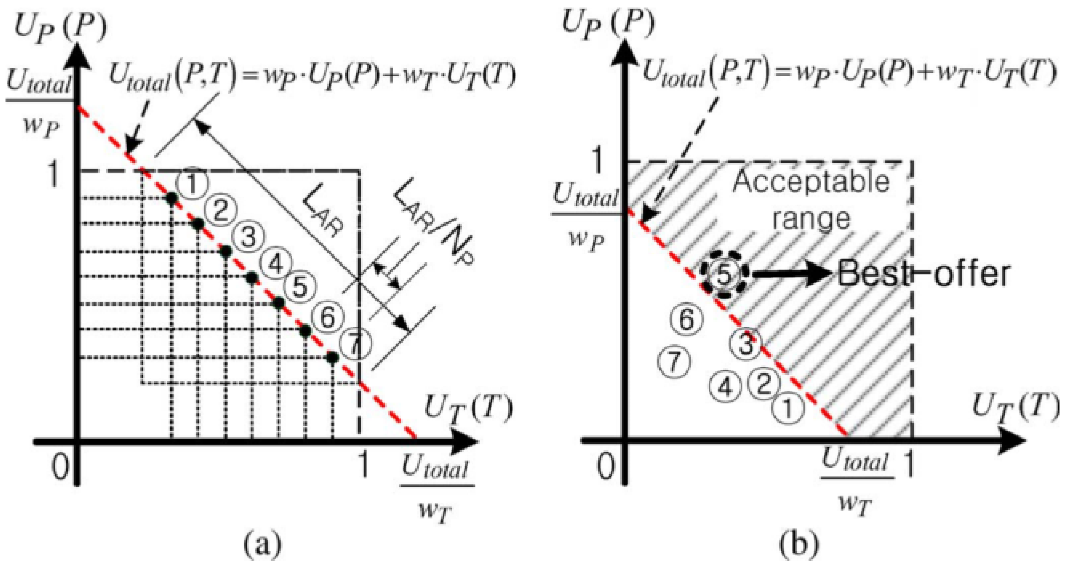


Figure 2.5: Generating (a) and evaluating (b) a “burst-7” proposal [6].

2.3.2 COCA

COCA, proposed in [4], uses a disruptive pricing formula. Before the auction, each provider predetermines its marginal price function, $P(x)$, a non-decreasing pricing function, based on utilisation. It represents the providers price.

The final price, paid by the user, *p*, is a function of the allocation γ and the submission time t_{sub}, as described in Equation 2.1. The allocation is the one that maximises the user utility. The utility is based on the parameters for each type of user, detailed in Section 2.1.4.B, (I, II, III), and the algorithm tries to optimise start and end times. It is assured that it cannot be negative and does not exceed the maximum capacity.

$$p(\gamma, t_{sub}) = \int_{t^-}^{t^+} \int_{U(t,t_{sub})}^{U(t,t_{sub})+\gamma(t)} P(x) \cdot Q dx dt \tag{2.1}$$

The values t^- and t^+ are the allocation start and end time, respectively. The function $U(t, t_{sub})$ represents the Utilisation Rate - quantity of resources allocated at t, observed on the instant t_{sub}. The value Q is the provider’s total capacity.

2.3.3 Ginseng

A bidding strategy is important because with a low bid the participants risk losing the auction, but with an overbid, they risk operating at a loss. The system Ginseng, presented in [20], provides a strategy for the users to decide the bid price and the maximal memory quantity.

The decision of the bid price (assuming the amount is determined) depends on the user valuation function, that maps, for each amount, the user's gain. If it is monotonically rising (e.g. $V(100MB) < V(200MB)$), the best bid is calculated by Formula 2.2, where $base$ represents the user minimum required amount. On the other hand, if the function is not monotonically rising, the users can define forbidden ranges for the intervals that are not economically efficient.

$$BestBidPrice = \frac{V(base + amount) - V(base)}{amount} \tag{2.2}$$

To decide the maximal memory quantity. The first step is finding the lowest price the guest can offer and still have a chance of getting any memory at all - obtained by evaluating the ten most recent borderline bids, published by the provider. In case of a concave valuation function, the memory is calculated by solving the Equation 2.3 in order to $maxMem$. In the other case, the user will split the valuation function into concave intervals. If multiple quantities are optimal, the user chooses the highest bid.

$$\frac{V(base + maxMem) - V(base)}{maxMem} = minPrice \tag{2.3}$$

2.3.4 Privacy Preservation System

The framework described in [27] addresses the privacy preservation issue in two independent modules: CloudPdr and CryptPdr. The users send encrypted bids to a cloud provider (CloudPdr) which runs the privacy-preserving auction, with the intervention of a crypto-service provider (CryptPdr), whose role is to enable private-preserving computations in the auction.

This protocol is based on two cryptographic concepts: construction of a garbled circuit and homomorphic encryption operations. This circuit allows two mistrusting parties to evaluate a function over their private inputs without the presence of a trusted third party. A typical problem solved by a garbled circuit is "Without revealing their actual wealth, who is richer? Alice or Bob?". Homomorphic encryption is a form of encryption that allows computations to be carried out on cipher text, thus generating an encrypted result which, when decrypted, matches the result of operations performed on the plain-text.

The protocol is the following: first, each bidder encrypts its bid using the public key of CryptPdr, and submits it to CloudPdr. Next, CloudPdr collects the encrypted bids and cooperates with CryptPdr to share the bids using secret sharing. CryptPdr generates a garbled circuit and sends it to CloudPdr.

Finally, CloudPdr executes the circuit with the encrypted bids, obtaining the auction result in plain-text.

2.4 Commercial Systems

We present the systems implemented by AWS, Google Cloud and Microsoft Azure.

2.4.1 AWS EC2

One of the first commercial cloud auctions is AWS EC2 Spot Instances and appeared in 2009. The winning user will pay the spot market price. It is calculated considering the price history for that configuration, and it is regularly updated. The market updates periodically (i.e., call market), and the bids stay in the system until fulfilled or cancelled by the user. The user can express its valuation by submitting his maximum bid price. Regarding the time, there are two policies: unlimited and predefined.

In the **unlimited** policy, social welfare is not assured as the user may lose the instance at any time if there is an increase in the spot market price.

In the **predefined** duration, there is a time guarantee where the instance is not lost. The user can choose from 1 to 6 hours. Naturally, the hourly price is higher compared to the unlimited policy pricing.

Table 2.1 allows us to understand the different pricing models and their variations better. These prices are available online and were recorded on 19/09/2017 for the Europe (Ireland) location. The spot prices are updated periodically but, according to the provider, they never vary more than 5%. They represent three categories of instances: general purpose, compute optimised, and memory optimised, respectively.

Table 2.1: Hourly prices for different VM types, in AWS.

		Spot			**Reserved**	
Type	**On-Demand**	**Unlimited**	**Pred. (1 h.)**	**Pred. (6 h.)**	**No-up. (1 y.)**	**Full-up. (1 y.)**
m4.large	$0.111	$0.0192	$0.056	$0.072	$0.076	$0.071
c4.large	$0.113	$0.0262	$0.057	$0.073	$0.078	$0.073
r3.large	$0.185	$0.0212	$0.074	$0.096	$0.136	$0.114

In 2015, AWS introduced a bid management tool - Spot Fleet Instances, allowing the user to specify multiple launch configurations and instance types. This system acts as a bid agent and is crucial in parallel and high-performance applications - the user can use the spot instances while maintaining high availability with no manual effort. It attempts to satisfy the user-defined target capacity if the spots are interrupted due to a change in spot prices.

Additionally, it is possible to choose a policy:

- **Lowest price**: spot instances come from the pool with the lowest price, consequently with more bids;

- **Diversified**: spot instances are distributed across all pools, decreasing the chance of an outbid - increased availability.

The work presented in [30] further develops this concept and details how to use spot instances to provision web applications, which are considered availability-critical.

2.4.2 Google Cloud

Also in 2015, Google launched Preemptible VM Instances [31] that run at a fixed lower price but are excluded from the SLA. Instances may not be available to start and may terminate abruptly. Additionally, every instance terminates after 24 hours. They are suitable for batch jobs and fault-tolerant applications. There is no auction, this type of VMs are delivered on-demand, and the billing is per minute (e.g. the user can terminate at any time). Table 2.2 allows us to measure the price reduction. These prices are available online and were recorded on 19/09/2017 for the Europe (London) location. They represent three categories of instances: general purpose, compute optimised, and memory optimised, respectively.

Table 2.2: Hourly prices for different VM types, in Google Cloud.

Type	On-Demand	Preemptible
n1-standard-1	$0.0612	$0.0123
n1-highcpu-4	$0.1824	$0.0366
n1-highmem-4	$0.3046	$0.061

2.4.3 Microsoft Azure

In 2017, Microsoft Azure implemented a ”low-priority” system very similar to Google Preemptible VM. They represent the spare capacity that exists in each region, for a significantly reduced price. The amount of spare capacity varies by region and VM size according to multiple factors, including the day of the week, time of day, and demand.

Similarly to Google, the VMs may not be able to start or may be preempted due to higher priority allocations (full-price).

Only a particular type of workloads benefit from this pricing model, namely:

- Media processing and transcoding - Some users may have flexibility and can tolerate their output potentially taking longer to produce, in which case their jobs could be run using low-priority VMs, and they could pay less for their jobs;

- Rendering - Jobs are split into many tasks, with individual frames or even tiles of frames, being able to be executed in parallel. Jobs may consist of thousands of tasks;

- Testing - Requires large scale and has some flexibility in when it completes. Particularly well suited for large-scale regression and load testing.

2.4.4 Critical Analysis

The three commercial systems we described offer improvements regarding the static pricing model, but they still have flaws. The winning user pays the spot market price, not its bid. This approach has two downsides. It is not beneficial to the provider, mainly, when the demand is lower than the supply - the provider may impose a hidden minimum price. As suggested in [32], this price does not reflect the client bids and the way it is calculated is not entirely transparent.

The second problem is the user benefits on bidding higher than its true valuation. In this situation, the user wins the auction but pays only the current spot market price. In fact, the spot price increases but only affects the future allocations. AWS EC2 Spot Instances is not truthful.

2.5 Summary

In this section, we present four tables to summarise the classification of cloud-auction systems. Some entries were marked with '-' meaning the system does not focus on that aspect. Table 2.3 describes the resource types, negotiation, and time dimension. The bid representation language is represented in Table 2.4. Table 2.5 summarises the mutual goals, and Table 2.6 concludes with the SLA and provider goals.

Table 2.3: System classification summary (1/4).

		Negotiation		Time	
System	**Resources Type**	**Participants**	**Bidding Scheme**	**Reservation**	**Market**
[16]	VM	Double	Unique	Spot	Continuous
[17]	VM	Double	Multi-round	Spot	-
COCA [4]	VM	Double	Unique	Forward	Continuous
Ginseng [20]	Memory	Single	Multi-round	Spot	Call
PTN [6]	-	Single	Direct negotiation	Forward	Continuous
[19]	VM	-	-	-	-
CDA [23]	-	Double	Multi-round	Spot	Continuous
[24]	-	-	-	Forward	Continuous
[18]	VM	-	-	-	Continuous
OptiSpot [5]	Application	-	Unique	Spot	Continuous
AWS EC2 Spot	VM	Single	Unique	Spot	Call

Table 2.4: System classification summary (2/4).

	Bid Representation Language			
System	**Resources**	**Quality Constraints**	**Time Restrictions**	**Price**
[16]	Type, fixed	-	None	Total
[17]	Type, fixed	QoS metrics	None	Unit
COCA [4]	Type, range	Types of users	Range	Based on user
Ginseng [20]	Type, range	-	None	Unit
PTN [6]	-	-	Range	Total
[19]	Type, fixed	Energy consumption	Range	Based on time
CDA [23]	-	-	None	Total
[24]	Type, fixed	-	Range	Total
[18]	Type, fixed	-	None	Total
OptiSpot [5]	Type	User and provider specs.	Range, provider valid	-
AWS EC2 Spot	Type	-	None	Unit

Table 2.5: System classification summary (3/4).

	Goals - Mutual		
System	**Price**	**Truthfulness**	**Privacy Preservation**
[16]	Average	Yes (argument)	-
[17]	Average	Yes (formal)	-
COCA [4]	Formula (both)	Yes (formal)	-
Ginseng [20]	Compensation	-	-
PTN [6]	Negotiation	-	-
[19]	Formula (both)	Yes (formal)	-
CDA [23]	Average	-	-
[24]	Formula (both)	-	-
[18]	Formula (both)	Yes (formal)	-
OptiSpot [5]	Model	No	-
AWS EC2 Spot	Spot price	No	-

Table 2.6: System classification summary (4/4).

	Goals - SLA	Goals - Provider	
System	**Social Welfare**	**Energy Efficiency**	**Time-slot**
[16]	-	-	First
[17]	Order by QoS	-	First
COCA [4]	-	-	Maximise valuation
Ginseng [20]	Guaranteed base	-	First
PTN [6]	Can reject bid	-	Negotiable
[19]	-	Considers	First
CDA [23]	Belief function	-	First
[24]	-	-	First
[18]	-	-	First
OptiSpot [5]	Mapping	-	First
AWS EC2 Spot	History, VM terminates abruptly	-	First

3

FairCloud

Contents

In this section, we present FairCloud, a Cloud-Auction system. After analysing the state-of-the-art, we extracted the desirable properties. We present the FairCloud bid representation language, the entities and events, and detail the core auctioneer algorithms. Also, in this section, we classify FairCloud using our taxonomy. Finally, we present the main implementation details of FairCloud.

3.1 Desirable Properties

FairCloud has three types of entities: User, Auctioneer and Provider. Figure 3.1 gives an overview of the system architecture with the participants and their main iterations (bid and allocation results).

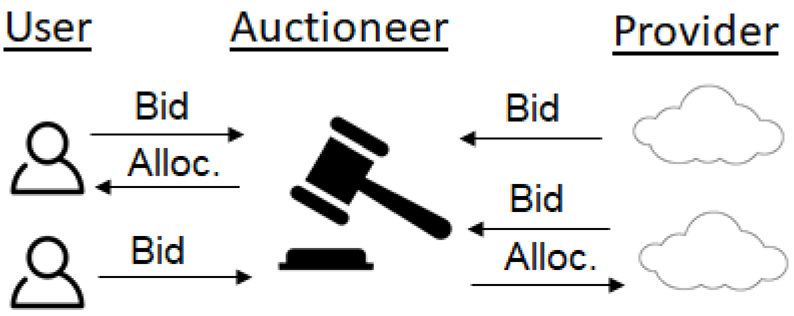


Figure 3.1: Overview of FairCloud use case.

It is crucial that a Cloud-Auctions system be continuous, combinatorial and truthful. A **continuous** auction is vital to reduce the user allocation time, as the bids are processed as soon as they arrive (see Section 2.1.3.B). Cloud applications are composed of multiple indispensable components (e.g. load-balancer, compute engine, storage). The bids should be **combinatorial** so that the participant can specify all those resources in one bid (see Section 2.1.4.A). Furthermore, some architectures require that all components be allocated (i.e., incomplete allocations are not desired). Therefore, a **roll-back** is necessary if all resources are not allocated. This operation is implemented in [16, 18, 24]. It should work similarly to a database transaction so the previous state is restored.

Truthfulness, studied in Section 2.1.5.B, is also fundamental. As the auctioneer does not know the participant's real valuation, truthfulness is the best method to prevent market manipulation, which can hurt the interests of other users and providers. Besides, truthfulness simplifies the strategic decision process for all participants, as their true valuations are the best strategies.

The charging period is the minimum interval the user can request and it will be one hour, as it is done on AWS EC2. Additionally, FairCloud should **facilitate the resource allocations**. Therefore, it must consider QoS requirements, in particular:

- VM degradation — partial service with only a minimum percentage of guaranteed resources;

- VM conversion — receive the preceding lower capacity machine (i.e., `large⇒medium`, `medium⇒small`, `small⇒micro` or `micro⇒nano`);
- Heuristics — strategies to maximise the number of allocations and their quality;
- Reputation mechanism — only based on internal metrics, used to sort the providers and allow the user to filter those under a minimum reputation threshold.

The specification of those features is done using a textual bid representation language.

3.2 Bid Representation Language

The bid representation language is composed by keywords, parameters and comments (starting with the symbol #). The following sub-sections (VM Types, Provider Bid, User Bid) detail the various parameters.

Listing 3.1 represents a provider bid and two user bids. The provider bid contains the capacity of 120 Million of Instructions per Second (MIPS) and 160 Megabyte (MB) of Random Access Memory (RAM), and its prices (e.g. each `medium` VM costs 0.612 currency units).

Particularly, user bid 0 requires a `demanding` profile, without conversions, with a minimum reputation of 50 and a timestamp of 0 microseconds. It requests a `nano` VM for 0.078 currency units. Similarly, user bid 1 requires a `relaxed` profile, without conversions, with a minimum reputation of 35 and a timestamp of 0 microseconds. It requests a `nano` VM for 0.077 currency units.

Listing 3.1: Bid Representation Language

```
#Example of the Bid Representation Language
#Used in section 3

#Provider Bids
new_provider 0 120 160
bid_provider 0
addvm_provider nano 0.077
addvm_provider small 0.306
addvm_provider large 1.529
addvm_provider micro 0.153
addvm_provider medium 0.612
submit_provider

#User Bids
bid_user 0 demanding noconversion 50 0
addvm_user nano 1 0.078
submit_user

```

```
bid_user 1 relaxed noconversion 35 0
addvm_user nano 1 0.077
submit_user
```

3.2.1 VM Types

The VM types were normalized to five general-purpose types: `nano`, `micro`, `small`, `medium`, and `large`. Each one specifies: weight (based on the computational power relations), CPU speed (measured in MIPS), RAM size (measured in MB), bandwidth and storage capacity. Table 3.1 details those types. The bandwidth and storage parameters are constant for all types.

Table 3.1: Detail of the VM types.

Type	Weight	MIPS	RAM (MB)
`nano`	1	75	100
`micro`	2	150	200
`small`	4	300	400
`medium`	8	600	800
`large`	16	1200	1600

3.2.2 Provider Bid

A provider bid contains the keywords (`new_provider`, `bid_provider`, `addvm_provider`, `submit_provider`) and the following parameters:

- **ID** Unique Identifier;
- **MIPS** Maximum available quantity;
- **RAM** Maximum available quantity, measured in MB;

For each VM type available:

- **VMType** One of the five types;
- **Price per hour** The minimum price requested, per unit.

3.2.3 User Bid

A user bid contains the keywords (`bid_user`, `addvm_user`, `submit_user`) and follows a similar structure:

- **ID** Unique Identifier;

- **Degradation Profile** One of the following: `demanding` (100% resources), `restricted` (80%), `relaxed` (60%). It represents the minimum partial service required;
- **Conversion** Indicates if the user accepts conversions;
- **Reputation** The minimum desired of the provider — number from 0 to 100.
- **Timestamp** When the bid should be visible to the auctioneer, measured in microseconds;

For each VM type requested:

- **VMType** As described above;
- **Amount**;
- **Price per hour** The maximum price accepted, per unit.

3.3 Entities and Events

The FairCloud participants communicate through asynchronous events. An event specifies the target (can be self), the delay (can be 0), the operation ID code, and the parameters. Figure 3.2 summarises, through a Unified Modelling Language (UML) sequence diagram, the protocol of the interactions involved among the different participants.

The participants start by registering in the auction using their Unique Identifier. Later, the auctioneer sends an ”AuctionStart” event, waits for the bids and sends ”Bid Acknowledges”. Next, the allocation algorithm executes, and the participants are notified by the ”Allocation Publication” event. According to the results, each user creates the requested VM in the corresponding provider, sends the jobs and waits for their return.

Additionally, the provider notifies the auctioneer that the execution ended. The notification allows the auctioneer to release the resources from the bid, allowing a new allocation.

The auctioneer is continuous. To implement this property, it sends self ”AuctionRepeat” events with a delay of three minutes. This entity is also responsible for evaluating the algorithm by measuring all providers utilisation of CPU and RAM. The monitoring occurs every five minutes. These metrics are transparent to the provider and do not affect the algorithm.

3.4 Auctioneer Data Structures

FairCloud is supported by four data structures. The ”UserBidList” and ”ProviderBidList” are managed by the auctioneer and contain all the user and provider bids, respectively. When a new bid is placed, the

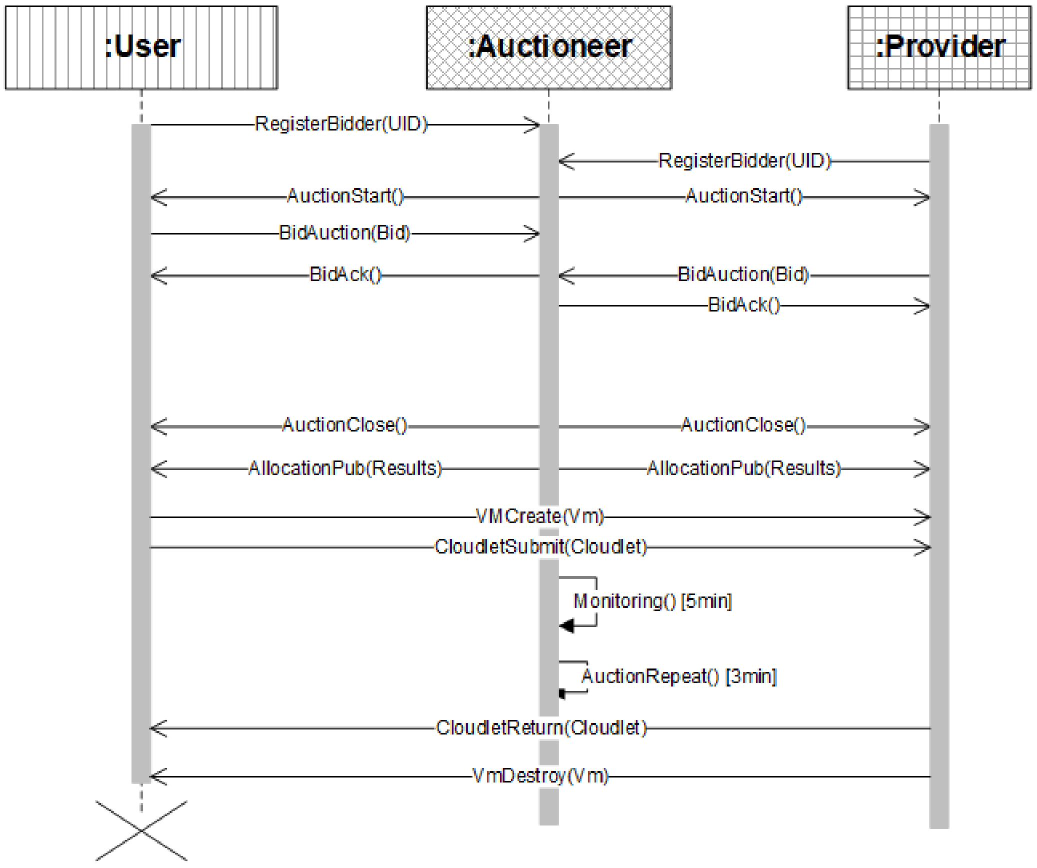


Figure 3.2: Sequence Diagram describing FairCloud entities and events.

auctioneer adds it to the corresponding list. Later, when the auction is in progress, the auctioneer sorts, iterates, and updates the bids in the lists.

The ”AllocationManager”, also managed by the auctioneer, maps, for each user, a list of ”AllocationEntry”. An ”AllocationEntry” contains the following information: provider ID, allocated VM, amount, profile, reputation score, and unit price. It is reset before each auction execution and committed at the end when the allocations are sent to the providers. If a user cannot be fully satisfied, its temporary allocations suffer a roll-back.

Lastly, the ”ReputationMap” maps, for each provider, a queue of 20 values, representing the last 20 scores awarded to that provider (i.e., quality for the allocations). This number was based on Ginseng [20] that records the ten most recent bids, but we doubled the number to provide a more accurate metric. It implements update and calculation operations.

3.5 Auctioneer Algorithms

FairCloud implements multiple algorithms. The Bid Sorting and Matching algorithm receives and matches the user and the provider bids, and invokes the Update Heuristics algorithm and the Auction Engine algorithm. After the allocations are decided, the Auction Engine calls the Assignment Protocol. Finally, we explain the mechanism and the reputation operations: Update and Calculation.

3.5.1 Bid Sorting and Matching

The Bid Sorting and Matching (Algorithm 3.1) is triggered in regular periods of three minutes. The value should be low enough to simulates a continuous market where bids are processed as soon as they arrive. However, a very small value would cause unnecessary algorithm executions, leading to wasted resources. The first step is to initialise the ”AllocationManager”. Then, it is divided into three phases: Bid Sorting, Matching, and Roll-back.

Bid Sorting Initially, the ”UserBidList” is ordered descending by bid density. Then, a join operation is made between ”ProviderBidList” and ”ReputationMap”. The ”ProviderBidList” is ordered ascending by $\frac{BidDensity}{Reputation}$. By using this ratio, if two providers have the same price, the one with more reputation is preferred.

The **bid density** is a measure of how much a participant bids per unit of allocation and it is given by Equation 3.1. The data structure ”VMType[]” represents all the VM types offered/requested in the bid, and its amount and unit price. The limit K represents the number of different VM types in the bid (i.e., size of the ”VMType[]”). In the ”UserBid”, the amount parameter is specified in the bid. However, on the ”ProviderBid”, the amount is obtained dynamically by dividing the provider total capacity for each VM capacity. Assuming a datacenter with 1500 MIPS of capacity and a `nano` VM with 75 MIPS, the amount is 20.

$$BidDensity(Bid) = \frac{\sum_{k=1}^{K} Bid.VMType[k].Amount \cdot Bid.VMType[k].UnitPrice}{\sqrt{\sum_{k=1}^{K} Bid.VMType[k].Amount \cdot weight(Bid.VMType[k])}} \tag{3.1}$$

By using this formula, in particular, the square root in the denominator, the users who request more resources and the providers with less capacity are prioritised. This way, the largest requests are allocated sooner as more capacity is available, leaving the remaining capacity for smaller requests that are easier to accommodate. This strategy reduces fragmentation, increasing energy efficiency. Furthermore, the small providers are prioritised, avoiding a scenario were leading providers could satisfy all requests, dominating the market. This bid density formula is also implemented in [16, 17].

Matching For each ”UserBid”, the algorithm selects the bids that have a timestamp greater or equal than the current time, and updates the heuristics. Then, the algorithm iterates the ”ProviderBidList”, focusing on ”ProviderBids” where the ”UserBid” minimum reputation is assured, and the Auction Engine executes.

Roll-back FairCloud should be able to roll-back the temporary allocations. First, the roll-back algorithm checks if it is necessary and, if it is, the allocations regarding that broker on the ”AllocationManager” are cleared. In the other case, the allocations are committed and sent to the participants when the auction ends. The reputation is only updated and committed in this step.

Algorithm 3.1: Bid Sorting and Matching.

```
input  : UserBidList, ProviderBidList, ReputationMap
output: UserBidList, ProviderBidList, ReputationMap
AllocationManager ← Initialise;
UserBidList.OrderDescendingBy(BidDensity);
ProviderBidList.Join (ReputationMap);
ProviderBidList.OrderAscendingBy($\frac{BidDensity}{Reputation}$));
foreach UserBid ub in UserBidList do
    if ub.timestamp $\geq$ Now() then
        UpdateHeuristics();
        foreach ProviderBid pb in ProviderBidList do
            if ub.Reputation $\leq$ Reputation.Calculate(pb.uid) then
                auctionEngine(ub, pb);
        rollback();
```

3.5.2 Heuristics

FairCloud should be able to degrade bids to increase the number of allocations, but only when necessary. To avoid compromising the algorithm’s execution speed, we use heuristics. They are a practical method not guaranteed to be optimal or perfect but consume far less computational resources than optimal approaches.

Based on the current available capacity and the requested capacity (i.e., ”UserBids” waiting to be allocated), Algorithm 3.2 updates two heuristics. They represent if and what degradations/conversion should be applied to ensure the maximum allocation.

Practical examples We demonstrate the algorithm by presenting two examples, complemented with the algorithm execution traces. The first lines represent the provider (capacity) and user bids (requested MIPS, profile, conversions). In the following lines, we present the total available and requests MIPS, which are used by Algorithm 3.2 and its output: profile heuristic and conversion heuristic. The last

Algorithm 3.2: Update Heuristics.

```
input  : UserBidList, ProviderBidList
output: ProfileHeuristic, ConversionHeuristic
TotalAvailableMips $\leftarrow$ SumAvailableMips(ProviderBidList);
TotalRequestedMips $\leftarrow$ SumRequestedMips(UserBidList);
if TotalAvailableMips $\geq$ TotalRequestedMips then
    ConversionHeuristic $\leftarrow$ False;
    ProfileHeuristic $\leftarrow$ Demanding;
else if TotalAvailableMips $\geq$ TotalRequestedMips $\cdot$ 0.8 then
    ConversionHeuristic $\leftarrow$ False;
    ProfileHeuristic $\leftarrow$ Restricted;
else if TotalAvailableMips $\geq$ TotalRequestedMips $\cdot$ 0.6 then
    ConversionHeuristic $\leftarrow$ False;
    ProfileHeuristic $\leftarrow$ Relaxed;
else
    ConversionHeuristic $\leftarrow$ True;
    ProfileHeuristic $\leftarrow$ Relaxed;
```

column shows the user that was allocated in that round. The last line summarises the remaining available/requested capacity after the execution. In the examples, we assumed that the bids are processed ascending by the user ID.

The Practical Example 1, presented in Table 3.2, uses the configuration from Listing 3.1. Assuming a total available capacity of 120 MIPS and two requests for 75 MIPS each, performing a total of 150 MIPS requested. User 0 requires a `demanding` profile and User 1 requires a `relaxed`. Neither participant allow conversions.

In the first call, Algorithm 3.2 would suggest a degradation to `restricted`. However, the first request did not allow that degradation, so the maximum capacity was assigned. In the following heuristic update round, the total capacity was now 45 MIPS (120 minus 75) and the request capacity 75 MIPS. The profile heuristic output was `relaxed`, so the second request was assigned only 60% of the requested capacity.

Table 3.2: Heuristics practical example 1.

Bids					
Provider0(120 MIPS), User0(75 MIPS, `Demanding`, `noconv.`), User1(75 MIPS, `Relaxed`, `noconv.`)					
Round	**Available MIPS**	**Requested MIPS**	**ProfileHeuristic**	**ConversionHeuristic**	**Allocation**
1	120	150	`Restricted`	False	User0
2	45	75	`Relaxed`	False	User1
-	0	0	-	-	-

The second example, presented in Table 3.3, contains three providers and six users with a variety of profiles and conversions acceptance. The initial supply is 60% of the demand. Therefore, the suggested profile is `relaxed`. In round 3, due to previous user's profile restrictions, the conversion heuristic is updated to true. In round 5, it is set back to false.

In these examples, we observe that the heuristic algorithm is optimal (i.e., all requests are satisfied, and there is no remaining available capacity, which could be used to improve those requests' quality).

Table 3.3: Heuristics practical example 2.

Bids					

Provider0(105 MIPS), Provider1(210 MIPS), Provider2(315 MIPS)
User0(225 MIPS, `Demanding`, `noconv.`), User1(75 MIPS, `Demanding`, `conv.`)
User2(150 MIPS, `Restricted`, `conv.`), User3(150 MIPS, `Restricted`, `conv.`)
User4(75 MIPS, `Relaxed`, `conv.`), User5(225 MIPS, `Relaxed`, `conv.`)					

Round	**Available MIPS**	**Requested MIPS**	**ProfileHeuristic**	**ConversionHeuristic**	**Allocation**
1	630	900	`Relaxed`	False	User0
2	405	675	`Relaxed`	False	User1
3	330	600	`Relaxed`	True	User2
4	255	450	`Relaxed`	True	User3
5	180	300	`Relaxed`	False	User4
6	135	225	`Relaxed`	False	User5
-	0	0	-	-	-

3.5.3 Auction Engine

First, Algorithm 3.3 iterates the requested VM list and checks if the user price is higher than the provider price and if the requested amount is positive. As we saw in the practical example, the profile is the maximum between the suggested by the heuristic and the user requested profile. The available amount is calculated by dividing the total capacity and each VM requested capacity, considering the degradation factor. Particularly, to calculate the conversion assign amount, we considers the profile value as 0.5 because each VM computational power is half of the following type. In other words, a converted type consumes half of the resource of the original type.

Naturally, the assigned amount is the minimum between the requested and available amount. Finally, if the user accepts conversions and the heuristic suggest a conversion, an ”AssignVMsConversion” takes place. Otherwise, an ”AssignVMs” takes place.

3.5.4 Assignments

The assignment protocol (i.e. Algorithms ”AssignVMsConversion” and ”AssignVMs”) is composed of two steps: price calculation and reputation award. After the last step, the information regarding the allocation is added in the ”AllocationManager”.

Algorithm 3.3: Auction Engine.

```
input : UserBid, ProviderBid
foreach Vm vm in UserBid.getRequestedVms() do
    if vm.price $\geq$ ProviderBid.PriceFor(vm) then
        if UserBid.getRequestedAmount(vm) $= 0$ then
            continue;
        AvailableMips $\leftarrow$ ProviderBid.getAvailableMips();
        AvailableRam $\leftarrow$ ProviderBid.getAvailableRam();
        Profile $\leftarrow$ Max(UserBid.Profile, ProfileHeuristic);
        AvailableAmount $\leftarrow$ Min(AvailableMips / (vm.mips $\cdot$ Profile.value), AvailableRam / (vm.ram $\cdot$
          Profile.value));
        AssignAmount $\leftarrow$ Min(UserBid.getRequestedAmount(vm), AvailableAmount) ;
        AvailableAmountConversion $\leftarrow$ Min(AvailableMips / (vm.mips $\cdot$ 0.5), AvailableRam /
          (vm.ram $\cdot$ 0.5));
        AssignAmountConversion $\leftarrow$ Min(UserBid.getRequestAmount(vm),
          AvailableAmountConversion) ;
        if ConversionHeuristic and UserBid.ConversionAllowed and AssignAmountConversion $\geq 0$
          then
            AssignVMsConversion(UserBid, vm, ProviderBid, AssignAmountConversion );
        else if AssignAmount $\geq 0$ then
            AssignVMs(UserBid, vm, ProviderBid, AssignAmount, Profile );
```

Price The first step in the assignment is to determine the final price. The final unit price is calculated with Formula 3.2.

$$Average(UserPrice, ProviderPrice) \cdot degradationFactor \cdot compensation \tag{3.2}$$

The degradation factor is the ratio of resources allocated and it is calculated before the allocation. The discount was introduced in [29, 33] and it is a compensation for the users with a performance degradation. The values of the partial utility matrix, presented in Table 3.4, were also obtained in that work. The notation "-" marks combinations not allowed. Consider the following example: the user profile is `demanding` and the provider (i.e., the assigned profile) offered a `conversion`. The degradation will be 0.5 (a `conversion` guarantees half of the resources) and the compensation 0.6 (obtained from the matrix).

Table 3.4: Partial Utility Matrix.

Profile \ User	`Demanding`	`Restricted`	`Relaxed`
`Demanding`	1	1	1
`Restricted`	-	1	1
`Relaxed`	-	-	0.9
`Conversion`	0.6	0.8	0.9

Reputation The second step is to award the reputation score. It depends on the offered quality: `demanding` - 1, `restricted` - 0.98, `relaxed` - 0.95, and `conversion` - 0.93. We chose these values considering the degradation factors (i.e., 100%, 80%, 60%, 50%).

3.5.5 Reputation Update and Calculation

The roll-back algorithm calls the update reputation, Algorithm 3.4. The reputation calculation, Algorithm 3.5 is needed to sort the bids.

The first algorithm, update reputation, removes the oldest value of the queue, to assure it always has the 20 most recent scores. Then, it adds the new score. For convenience, it is possible to add multiple scores in one algorithm call, using the parameter Multiplier.

Algorithm 3.4: Update Reputation.

```
input  : ReputationMap, Provider, Value, Multiplier
output: ReputationMap
for i ← 0 to Multiplier do
    if ReputationMap.Get(Provider).IsFull() then
        ReputationMap.Get(Provider).RemoveFirst();
    ReputationMap.Get(Provider).Add(Value);
```

The reputation calculation algorithm describes how to obtain the final reputation value by multiplying each partial value.

Algorithm 3.5: Calculate Reputation.

```
input  : ReputationMap, Provider
output: Value
Value ← 100;
foreach Value v in ReputationMap.Get(Provider) do
    Value ← Value · v;
```

3.6 Classification using our Taxonomy

To summarise the architecture, we will assess our system according to the taxonomy proposed in Chapter 2. First, FairCloud trades general-purpose VMs of five predefined types: `nano`, `micro`, `small`, `medium`, and `large`.

It is a double auction, and the bidding scheme is unique (the bid stays in the system until it is allocated). Regarding the time dimension: the auctioneer makes spot reservations and operates in a continuous market.

The bid representation language supports the specification of the requested amount for each type. However, the provider can only specify its total capacity, regarding MIPS and RAM. FairCloud supports a variety of quality parameters: degradation profile (i.e., `demanding`, `restricted`, `relaxed`), conversions and desired minimum reputation. The only time restriction is the timestamp representing when the bid is visible to the auctioneer. The user and provider price is per unit per hour.

The equilibrium price is obtained by multiplying the user and provider average prices, the real amount of allocated resources and the compensation (depending on the user quality profile). We can argue that our system is truthful because if a participant bids different from its true valuation its position in the priority list will decrease. FairCloud does not consider privacy preservation. Regarding the SLA, FairCloud implements an internal reputation mechanism to order the providers and filter those under a minimum threshold. Furthermore, the reputation does not depend on metrics collected by the provider that could be easily forged. Finally, by implementing an auction, we can reduce the final price paid per VM increasing the utilisation and the energy efficiency. The algorithm always chooses the first available spot.

3.6.1 Scheduling Assessment

FairCloud is a cloud scheduler. It is a global and dynamic system, but it can also be static if we treat each round individually, because we know the currently available and requested capacities. In this scenario, it is suboptimal, and we use a heuristic with the goal of maximising the number of allocations while minimising the average request degradation.

Detailing the dynamic properties:

- Distributed, Cooperative — The allocation is divided into multiple hosts, and all cooperate to allocate the maximum resources possible;
- Non-adaptive — The algorithm does not modify itself;
- Load Balancing — Partially implemented, the allocation always goes to the first-fit, but the reputation might change the provider's order (i.e., the first provider changes);
- One-time assignment — The scheduling decision is made once.

3.7 Implementation Details

The FairCloud simulation has three layers: CloudSim, CloudAuctions (extension), and bid representation language.

CloudSim is a simulator that is widely used in works related to cloud energy efficiency, work-flows, scalability and pricing policies [34, 35]. This layer offers the Application Programming Interface (API) for VM management and for running the user cloudlets. A cloudlet represents the jobs to be executed. It also provides metrics (e.g. utilisation, energy consumption, profit, latency). There are many extensions to CloudSim developed by third parties, and it can be made parallelised and distributed [36].

We used the extension CloudAuctions, proposed, and used in [16]. This extension was updated to the latest CloudSim version, fine-tuned, and new functionalities were implemented (i.e. FairCloud algorithms). Finally, a bid representation language was created, to support the user and provider bid API. Figure 3.3 summarizes these layers.

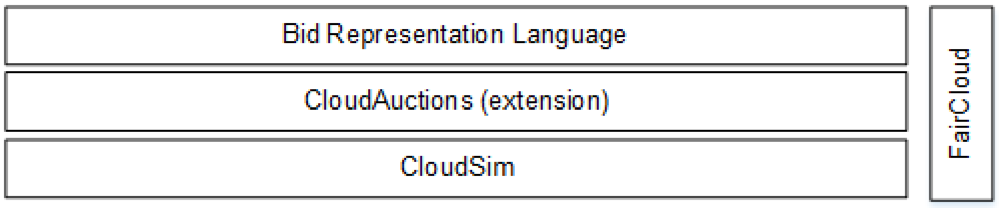


Figure 3.3: FairCloud layers.

3.7.1 Code Organization (Packages)

The UML package diagram represented in Figure 3.4 helps us understand the organization and the functionality of the system. We used a colour code grey, blue, green for representing the packages not changed, partially modified (e.g. new method) or new/totally reformulated, respectively.

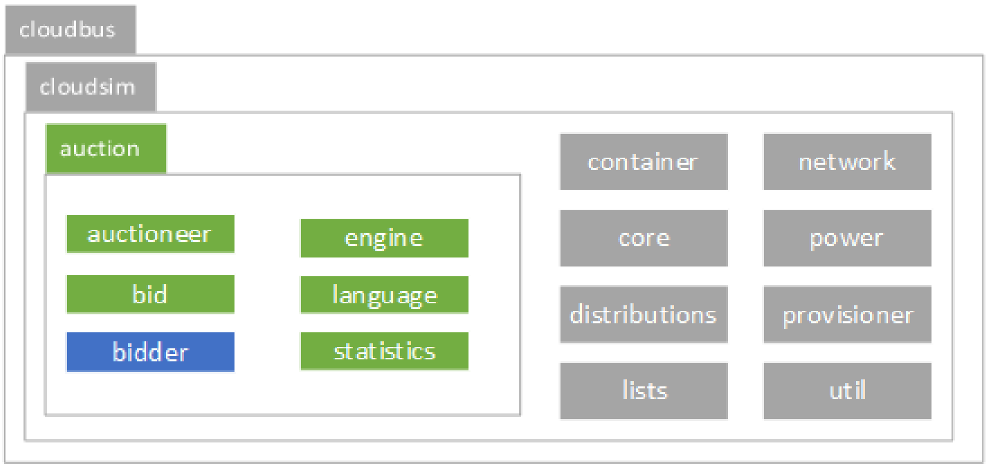


Figure 3.4: Implementation packages.

First, we will detail the auction package. The auctioneer contains all the classes to support the auction algorithms (e.g. ”AllocationManager”, ”NormComparator”). The bid package contains both the user and provider bids. The package engine comprises the particular auction strategies. This separation is particularly useful in the evaluation phase when multiple algorithms will be implemented but with shared functionality (e.g. communication, monitoring).

The language package contains the enumerates ”VMSize” and ”Profile”, and a configuration class. In this class, it is possible to specify, among others, the VM types. Finally, the statistics package deals with the reputation and the record of the metrics. This package is also useful for the evaluation.

A container is a virtual environment that provides a layer of isolation between workloads, and they are handled in the containers package. The core package accommodates the simulator’s engine (e.g. event queues, entities, shut-down process). The network package supports graph operations and communications (e.g. a network packet). The power package aggregates different power consumption models, with the resource usage as input, (e.g. linear, cubic, equal to IBM X3250). Here, we see that we can customise our providers with a consumption model. Perhaps, the most relevant package is the provisioner — it represents the hardware suppliers: MIPS, RAM, and bandwidth. The MIPS provisioner is denoted processing element. Finally, the util package is used to measure time and to read workloads.

3.7.2 Classes

In this section, we detail the core classes. We can make a distinction based on their semantic: bid-related and the simulator entities. CloudSim uses the terminology ”datacenter broker” and ”datacenter”, to represent the user and provider, respectively.

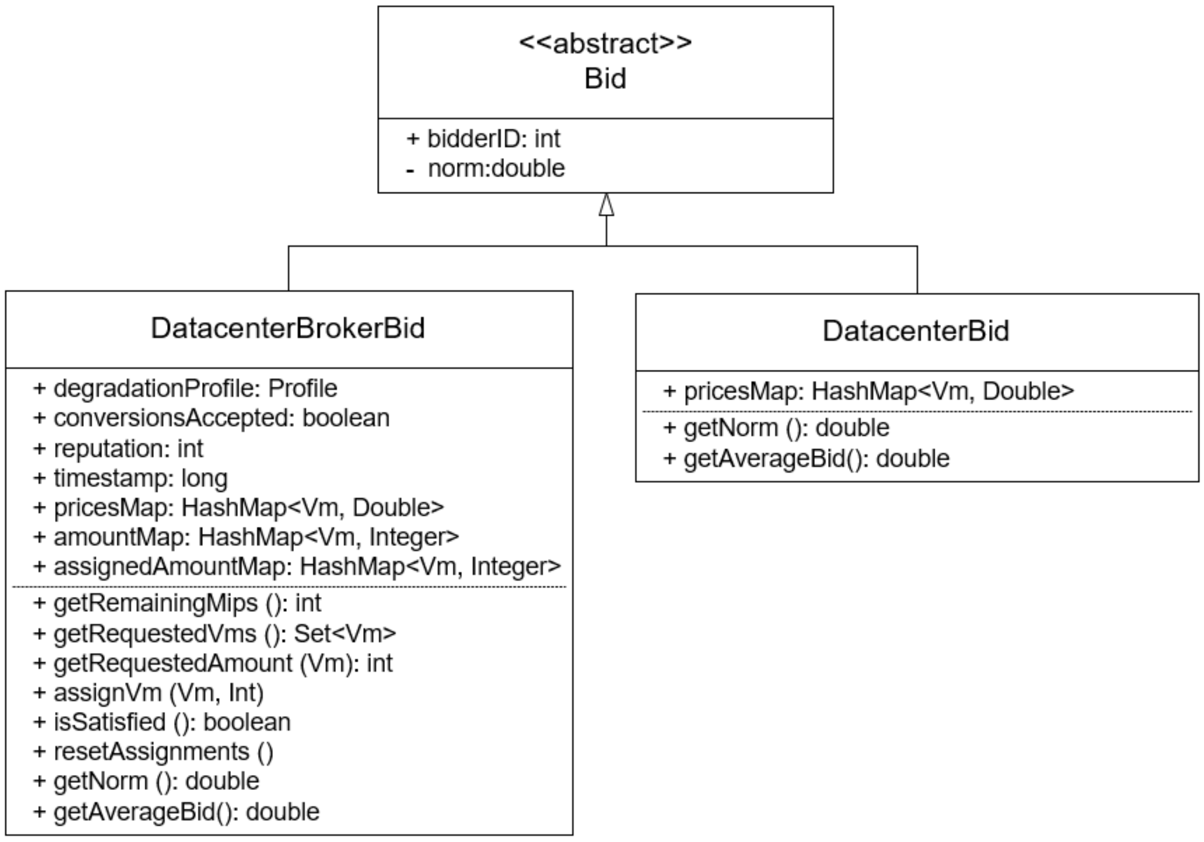
3.7.2.A Bid-Related

The bid-related classes are represented in Figure 3.5. The ”DatacenterBrokerBid” contains the bid representation language parameters and three data structures: ”pricesMap”, ”amountMap”, and ”assignedAmountMap”. They store the offered price for each VM, the total offered amount and the assigned amount. The methods operate with these maps, particularly by comparing the amount and the assigned amount. The ”getNorm” method contains a caching mechanism. The normal value is only calculated once, and its value is stored in the ”norm” attribute.

The ”DatacenterBid” is very similar. It contains the requested prices map and the methods to calculate the normal and the average bid.

3.7.2.B Simulator Entities

We will split this section into Auctioneer, and Datacenter and DatacenterBroker. However, all the three classes inherit from ”SimEntity”. It contains the entity name and id. It is possible to change the start and shut-down behaviour by extending the ”startEntity” and the ”shutdownEntity” methods.



<<abstract>>
Bid
+ bidderID: int
- norm:double
DatacenterBrokerBid
+ degradationProfile: Profile
+ conversionsAccepted: boolean
+ reputation: int
+ timestamp: long
+ pricesMap: HashMap<Vm, Double>
+ amountMap: HashMap<Vm, Integer>
+ assignedAmountMap: HashMap<Vm, Integer>
+ getRemainingMips (): int
+ getRequestedVms (): Set<Vm>
+ getRequestedAmount (Vm): int
+ assignVm (Vm, Int)
+ isSatisfied (): boolean
+ resetAssignments ()
+ getNorm (): double
+ getAverageBid(): double
DatacenterBid
+ pricesMap: HashMap<Vm, Double>
+ getNorm (): double
+ getAverageBid(): double

Figure 3.5: Class diagram with the core Bid-related classes.

"ProcessEvent" allows defining the action, depending on the incoming event (e.g. auction start, bid, auction end). "Send" is used to send an event.

Auctioneer The UML class diagram, in Figure 3.6, details the "CoreAuctioneer" class. This class contains the allocation manager, the broker bid list, and the datacenter bid list. The first eight methods handle the registering, bidding, and allocation publication functionality. "StopCondition" is used to evaluate when to stop the system. In a real situation, FairCloud would run continuously, but for experimental purposes, the algorithm stops once all bids were assigned or the simulation time passed three hours. The restore method releases the datacenter resources and measures the execution time. The monitoring method iterates the chain datacenter, host, VM, and calculates the CPU utilisation. Finally, the "rollback" method checks if the datacenter broker is satisfied. If it is, it commits the reputation. Otherwise, it restores the datacenter's resources, deletes the assignments in the allocation manager, and resets the bid assignments.

We implemented three engines: "faircloud", "cloudauctions", and "aws.ondemand". We will focus on the implementation on the first. The two attributes are the heuristics: profile and conversion. The methods regarding the auction were implemented following the algorithms described in Section 3.5. ”RecordProvidersOrder” iterates the ”datacenterBidList” and stores their order in the ”Stats” instance. Lastly, the repeat method sends an event with a three-minute delay to repeat the auction, unless the stop condition is positive.

The core auctioneer contains an ”AllocationManager” that contains multiple ”AllocationEntry” instances.

Datacenter and DatacenterBroker The ”DatacenterBroker” contains a ”submitCloudletList” method to define the cloudlets to be run. FairCloud extended it by creating a ”BidderDatacenterBroker”. The datacenter is composed by hosts. A host contains the hardware specifications: storage, RAM, bandwidth, and processing elements. It contains multiple VMs. Similarly, the ”Datacenter” was extended by creating a ”BidderDatacenter” that contains ”availableMips” and ”availableRam”. Those values are not redundant with the sum of hosts. They also consider the bids that were allocated in the auctioneer but not yet committed. Both entities implement a method to submit a bid (i.e., to be invoked by the launcher) and a method to bid in the auction. They contain an ”AuctionAgent” instance that handles all the incoming events, through the ”processAuctionEvent” method.

3.7.3 Launcher

Beside unit testing, CloudSim contains application examples for each of the packages described in Section 3.7.1. Following those applications’ structure, we developed a launcher to pre-process, interpret the bid representation language, and run the simulation.

Pre-processing consists of removing unnecessary spaces in the text file and normalising all the commas in decimal values to dots. Then, the launcher creates an ”Auctioneer”. Depending on the keywords, different actions are taken:

- **new_provider**: Create a ”BidderDatacenter”, including its ”Host” and the provisioners;
- **bid_provider**: Create a ”DatacenterBid”;
- **addvm_provider**: Update the bid by adding the new type and price;
- **submit_provider**: Submit the bid to the provider;
- **bid_user**: Create the ”BidderDatacenterBroker” and the ”DatacenterBrokerBid”;
- **addvm_user**: Update the bid by adding the new requested VM, amount and price;
- **submit_user**: Create a cloudlet for each requested VM with the full utilization model (i.e., all available resources are used). Submit the bid and the cloudlet list to the user.

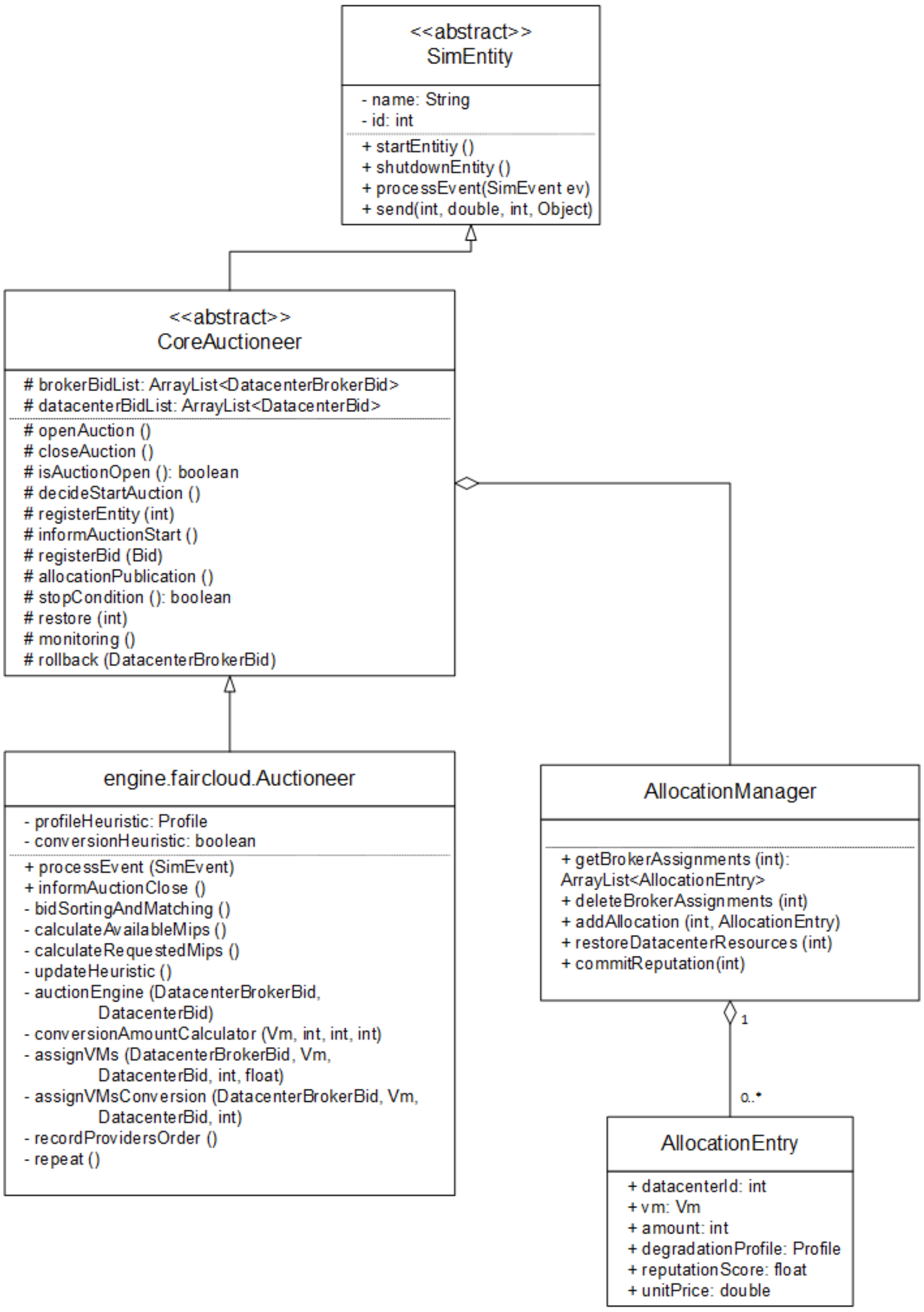

<<abstract>>
SimEntity
- name: String
- id: int
+ startEntitiy ()
+ shutdownEntity ()
+ processEvent(SimEvent ev)
+ send(int, double, int, Object)
<<abstract>>
CoreAuctioneer
# brokerBidList: ArrayList<DatacenterBrokerBid>
# datacenterBidList: ArrayList<DatacenterBid>
# openAuction ()
# closeAuction ()
# isAuctionOpen (): boolean
# decideStartAuction ()
# registerEntity (int)
# informAuctionStart ()
# registerBid (Bid)
# allocationPublication ()
# stopCondition (): boolean
# restore (int)
# monitoring ()
# rollback (DatacenterBrokerBid)
engine.faircloud.Auctioneer
- profileHeuristic: Profile
- conversionHeuristic: boolean
+ processEvent (SimEvent)
+ informAuctionClose ()
- bidSortingAndMatching ()
- calculateAvailableMips ()
- calculateRequestedMips ()
- updateHeuristic ()
- auctionEngine (DatacenterBrokerBid, DatacenterBid)
- conversionAmountCalculator (Vm, int, int, int)
- assignVMs (DatacenterBrokerBid, Vm, DatacenterBid, int, float)
- assignVMsConversion (DatacenterBrokerBid, Vm, DatacenterBid, int)
- recordProvidersOrder ()
- repeat ()
AllocationManager
+ getBrokerAssignments (int): ArrayList<AllocationEntry>
+ deleteBrokerAssignments (int)
+ addAllocation (int, AllocationEntry)
+ restoreDatacenterResources (int)
+ commitReputation(int)
1
0..*
AllocationEntry
+ datacenterId: int
+ vm: Vm
+ amount: int
+ degradationProfile: Profile
+ reputationScore: float
+ unitPrice: double

Figure 3.6: Class diagram with the SimEntities (1/2) — Auctioneer.

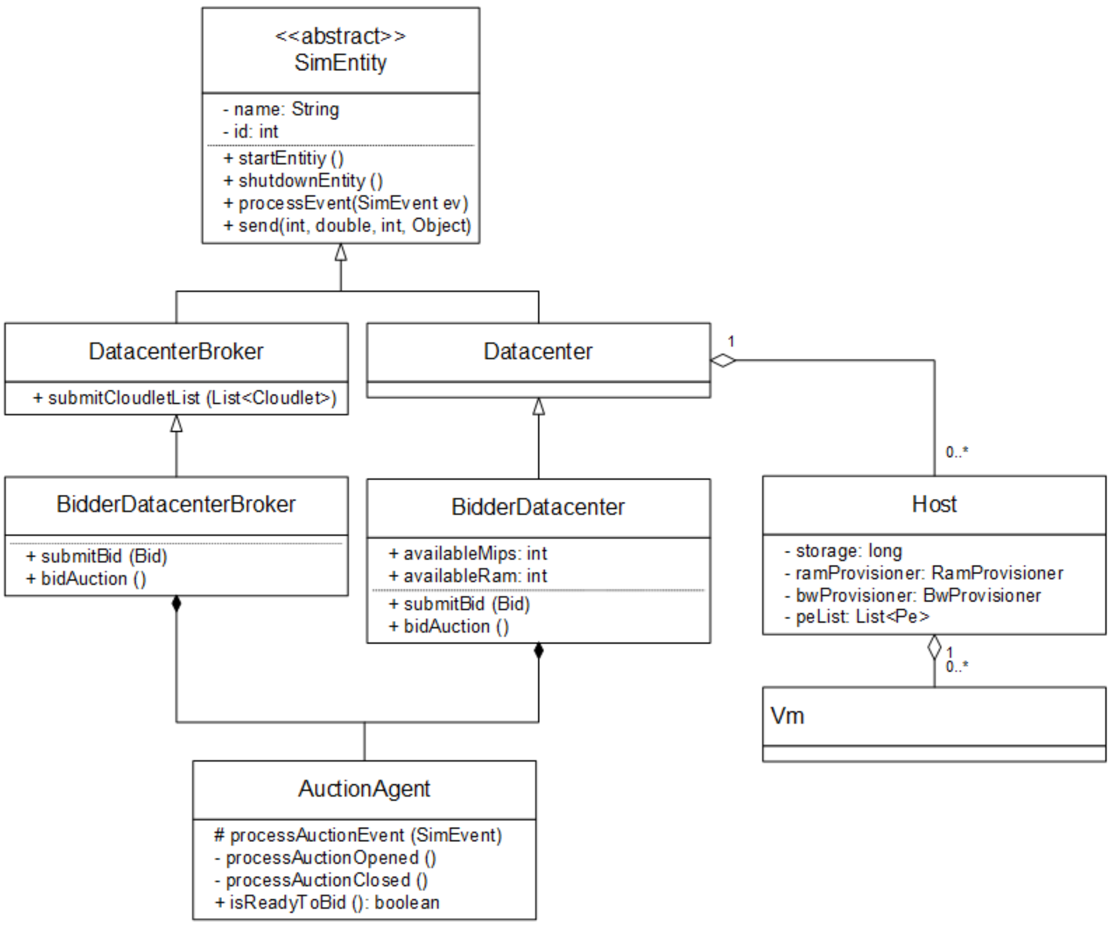


Figure 3.7: Class diagram with the SimEntities (2/2) — Datacenter and DatacenterBroker.

Finally, we initialise ”CloudSim” and invoke its the start and stop methods.

3.8 Summary

We started the architecture by presenting the desirable properties for a cloud-auction: continuous, truthful, combinatorial and considering QoS requirements, in particular, degradations, conversions, heuristics to maximise allocations and their quality, and a reputation mechanism. We also describe the bid representation language. FairCloud is supported by four data structures: ”UserBidList”, ”ProviderBidList”, ”AllocationManager”, and ”ReputationMap”. The entities are users, providers, and auctioneer.

The core algorithms are the bid sorting and matching, update heuristics, auction engine, and assignment protocol. The assignment protocol calculates the final price by multiplying the average (user and provider price), degradation factor and compensation. The roll-back is necessary if the allocation is not entirely satisfied. FairCloud also implements the reputation update and calculation.

To summarise the architecture, FairCloud was assessed considering our taxonomy.

Lastly, we described FairCloud implementation. It was implemented in CloudSim, using one of its extensions — CloudAuctions. The bid representation language was implemented in the superclass ”Bid” and on its sub-classes ”DatacenterBrokerBid” and ”DatacenterBid”. Our protocol is supported by entities that inherit from ”SimEntity”. In particular, the auctioneer is divided into two classes: ”CoreAuctioneer”, containing the communication, auction and monitoring functionality, and multiple ”Auctioneer”, with different auction engines. The user was implemented in the classes ”DatacenterBroker” and ”BidderDatacenterBroker”. The provider was implemented in the classes ”Datacenter” and ”BidderDatacenter”.

4 Evaluation

Contents

In this chapter, we evaluate FairCloud and compare it with other systems. Lastly, we state which type of participants benefit from using FairCloud and why.

To evaluate FairCloud, we used a subset of the Google Cluster Data (2011_2), described in [37, 38]. It is widely analysed and used in evaluations [18, 19, 39–42]. This data provides information regarding Machines, Jobs and Tasks, and Resource Usage. It represents 29 days' worth of cell information from May 2011.

Table 4.1 shows the tables we used and their schema.

Table 4.1: Google Cluster Data tables used and their schema.

Machine events	Task events
Timestamp	Timestamp
Machine ID	Job ID
Event type	Event type
Capacity: CPU	Scheduling class
Capacity: Memory	Priority
	Requested CPU
	Requested RAM

4.1 Dataset Details

We prepared eight datasets with different goals and characteristics. Table 4.2 summarizes their configurations.

Table 4.2: Datasets configuration summary.

	Providers	User Degradation Profile	Conversions	Min. Reputation	Price
1	9 Homogeneous	`Restricted`	No	0	Equal
2	9 Homogeneous	Variable	No	0	Equal
3	9 Homogeneous	Variable	No	0	Variable
4	9 Homogeneous	Variable	Yes	0	Variable
4b	9 Homogeneous	Variable	Yes	(Disabled)	Variable
5	9 Homogeneous	Variable	Yes	Variable	Variable
6	1 Homogeneous	Variable	Yes	0	Variable
7	100 Heterogeneous	Variable	Yes	0	Variable

4.1.1 Filtering and Normalisation

First, we selected the entries with the timestamp between 0 and 2 hours. The entries with timestamp 0, representing the entries that were already scheduled, were also included. The provider data was obtained from the Table `Machine events` and we selected the `Add` events, which represents the machines added to the cluster. The user data was gathered from the `Task events` Table, focusing on the entries in the `Submit` transition, representing tasks eligible for scheduling.

For confidentiality reasons, the information in the trace was obfuscated. In particular, the CPU and memory fields were rescaled to the interval 0 to 1. Without having the real values, we did the following mapping, regarding CPU and VM: [0, 0.0626[- `nano`, [0.0626, 0.125[- `micro`, [0.125, 0.25[- `small`, [0.25, 0.5[- `medium`, and [0.5, 1] - `large`.

The user bids were obtained by grouping the tasks by `Job ID`. Each job represents a user bid and each task the VMs requested in that user bid. The jobs containing more than 30 tasks were divided into groups of maximum 30 tasks. This method prevented the total requested capacity from being higher than the total available capacity.

4.1.2 Providers

The Providers configuration was obtained by diving the total capacity (from Table `Machine events`) by 1, 9, or 100. Datasets 1 to 5 contain nine homogeneous providers, simulating the well-known and equivalent cloud providers (e.g. AWS, Google Cluster, IBM, Microsoft, OVH). Dataset 6 simulates a single global provider. Finally, Dataset 7 represents not only the well-known providers but also shared infrastructures, research labs, universities, condominiums, and communities

Dataset 7 was generated by dividing the total capacity by 100 and multiplying by a factor from a Random Normal Distribution (AVG=1; STD=0.2). The standard deviation was chosen in a way to maximise the range of values, but assuring that all were positive. This way, the sum of the requests is still close to the original. Figure 4.1 show the distribution of the 100 factors.

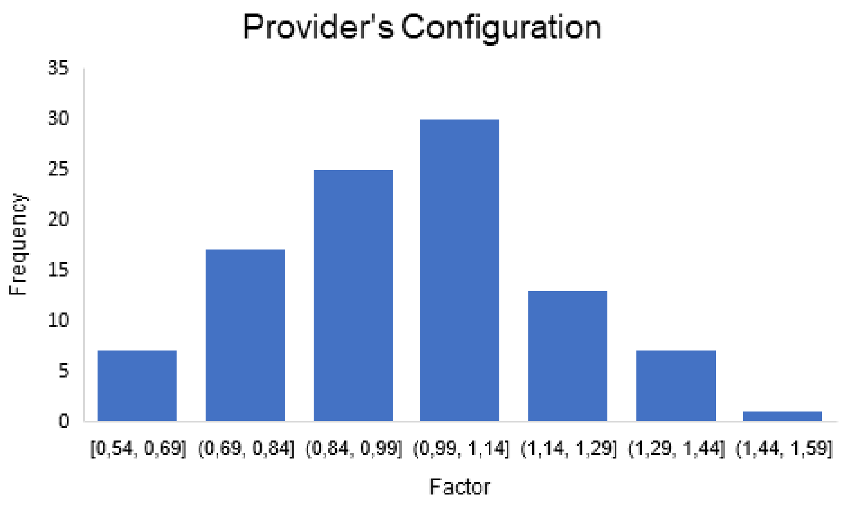


Figure 4.1: Provider's configuration factor.

4.1.3 Degradation Profile

In Dataset 1, the user profile is always `restricted`. The user profile is based on the priority field (integer from 0 to 11). In the Google Cluster Data, priority determines whether a task is scheduled on a machine. Each job contains a priority which was normalised by [40] to the following sets: production, middle, and gratis. The degradation profile assignment is the following: production - `demanding`; middle - `restricted`; gratis - `relaxed`.

Figure 4.2 (left) shows the distribution of three profiles. By analysing the user's dashboard, we concluded that there are 11475 users. Over the time, they request a sum of 208822 VMs.

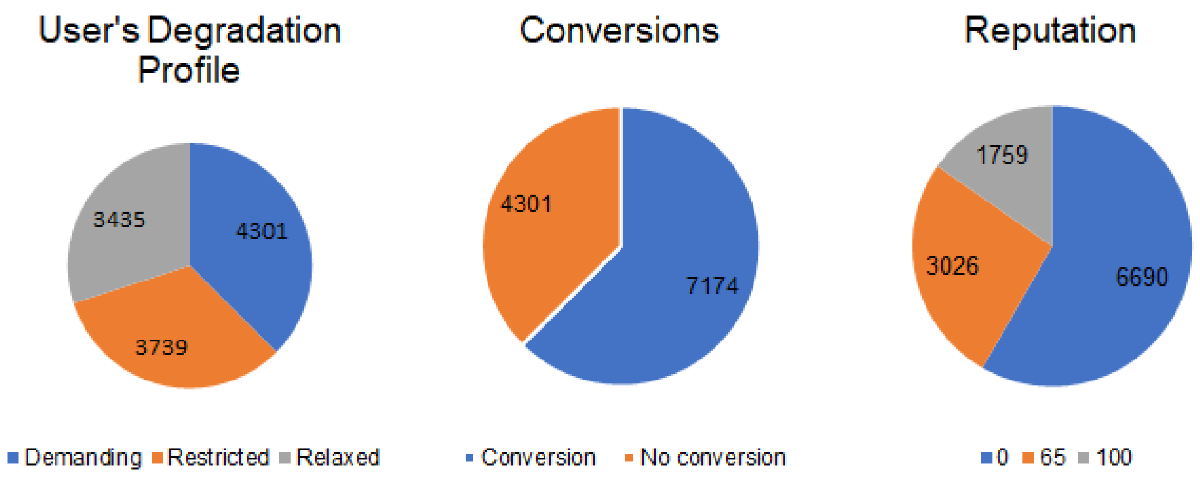


Figure 4.2: User's dashboard.

4.1.4 Conversions

Conversions can be allowed in the Datasets 4 to 7. The user bids resulting from jobs with a priority between 0 and 9 allow conversions. Figure 4.2 (middle) show the proportion of jobs allowing conversions.

4.1.5 Reputation

In the Datasets 1 to 4, 6 and 7, the minimum reputation required is 0. In Dataset 4b, the reputation is completely disabled (i.e. minimum and ordering). In Dataset 5, users require a minimum reputation. We used the field *scheduling class* that represents how latency-sensitive the job is. We did the following scheduling class to minimum reputation mapping: class [0,1] - 0; class [2] - 65; class [3] - 100.

Figure 4.2 (right) represents the minimum reputation required by the users.

4.1.6 Price

We define the base price as the AWS EC2 Spot Instances price, observed on 25/06/2017, for the Predefined 1h duration and the Europe (Ireland) location. We assigned the base prices 0.076, 0.153, 0.306, 0.0612, and 1.529 for the VMs `nano`, `micro`, `small`, `medium`, and `large`, respectively. The prices depend on the demand and are updated periodically on [43].

Depending on the dataset, the price is multiplied by a factor. The factor allows us to simulate participants that are willing to pay more than others.

In Datasets 1 and 2, the provider factor is 1, and the user factor is 1.5. The factors express that all the users pay 50% more than the provider requested.

In the remaining datasets, the provider factor is obtained from a Random Normal Distribution (AVG=1; STD=0.1) and the user from a Random Normal Distribution (AVG=1.5; STD=0.3).

Graphic 4.3 represents the different factors.

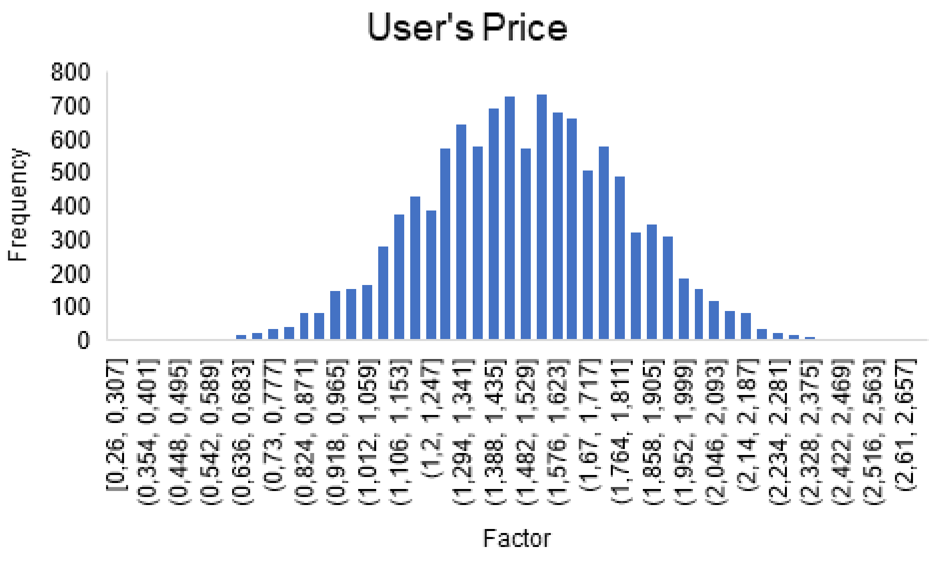


Figure 4.3: User's price factor.

4.2 Implementation in Pentaho DI

W prepared the datasets using Pentaho Data Integration - data extraction and transformation tool. In this section, we detail the implementation of the datasets previously described. We will focus only on the user transformation because it is far more complicated than the provider's transformation.

4.2.1 Filtering and Normalisation

The first step is a CSV file input where we specified the file location, the delimiter and the fields to read. Next, we filtered the `Submit` event, the positive CPU and RAM and the time-stamp under 7,200,000,000 microseconds (2 hours). The next step, number range, allowed to map intervals in values. Finally, a value mapper allowed to assign a Target Value based on a Source Value. It was used to assign the CPU, RAM, and the base price. These steps are represented in Figure 4.4.

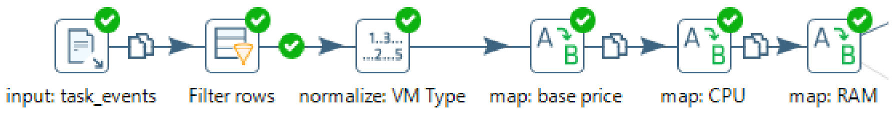


Figure 4.4: Pentaho DI — Filtering and Normalisation.

4.2.2 Configurations

Figure 4.5 represents the configuration steps.

Here, we also used number ranges to assign the profiles, conversions, and reputations. We can observe that some hops are grey, representing disabled hops. This way, it is possible to generate all datasets in a single transformation. The name of the step helps us to easily generate the datasets. For example, the name ”[2,7] map: profile” represents that it should be only activated to generate Datasets 2 to 7. The dummy step is only used to increase the readability of the transformation.

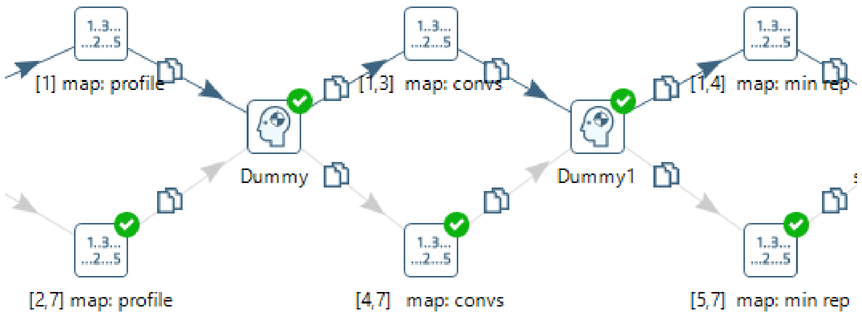


Figure 4.5: Pentaho DI — Configurations.

4.2.3 Grouping

The grouping transformation is represented in Figure 4.6.

Then, we group the tasks by job ID but, before, a sort rows is necessary. The next step is a user defined java method. It was necessary because none of the predefined steps could do the divisions. In the code snippet 4.7, we obtained the field values by using the ”get” operations and cloned them with the new task_count values, using "createResizedCopy". Finally, they were inserted into the stream using the "putRow". At this point, the maximum requests per user are 30. The add sequence step will assign the final user ID. The add constant value steps add the keywords to comply with the bid representation language. Finally, the sort step ensures that the rows are ordered by user ID and in the correct sequence (i.e., bid_user, addvm_user, submit_user).

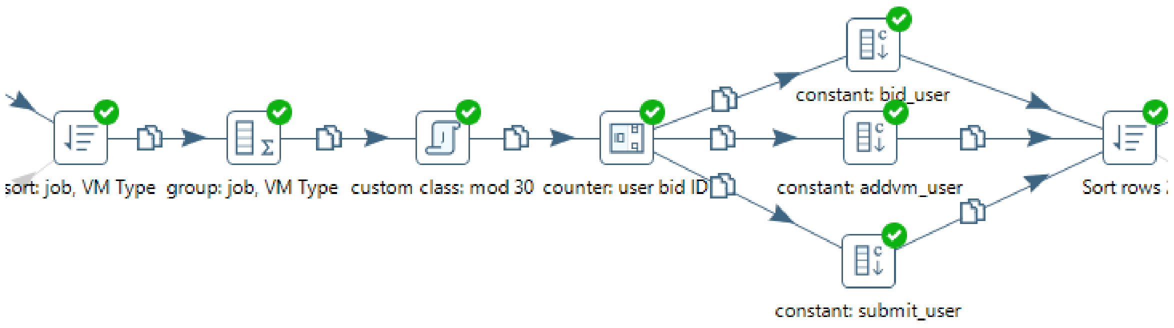


Figure 4.6: Pentaho DI — Grouping.

4.2.4 Prices and Output

The prices and output transformation is represented in Figure 4.8. In the step "[1,2] map: price factor", we map the factor 1.5 to all user bids. In the remaining datasets, the factors are loaded from a .CSV file and a inner join is performed. The final price is calculated multiplying the base price and the factor. The next step, user defined java method, concatenates the fields to form the final string. To avoid concurrent writes, the block until step finishes step waits until the provider's bids are written. The last step writes the fields a .txt file. Here, we can configure: appending, delimiters and headers.

4.3 Metrics

The evaluation metrics is divided into two categories. Global quality assesses the auction algorithm, and efficiency is focused on the system. Finally, we explain how we implemented them in FairCloud.

4.3.1 Global Quality

- **Average price** per VM determines if the auctions are beneficial compared to the traditional/static approach;
- The **allocation time** is the period from the bid being submitted to the system and the allocation starts. This is measured individually for each VM and in seconds. This metric is focused on users that need the resources as soon as possible (e.g. VM hosting a website);

```
public boolean processRow(StepMetaInterface smi, StepDataInterface sdi)
        throws KettleException {


    Object[] r = getRow();
    if (r == null) {
        setOutputDone();
        return false;
    }

    long task_count = get(Fields.In, "task_count").getInteger(r);
    long max=task_count/30;
    long remain=task_count%30;
    Object[] out_row;

    if (remain!=0){
        out_row = RowDataUtil.createResizedCopy(r, data.outputRowMeta.size());
        get(Fields.Out, "task_count").setValue(out_row, remain);
        putRow(data.outputRowMeta, out_row);
    }

    for (int i=0;i<max;i++){
        out_row = RowDataUtil.createResizedCopy(r, data.outputRowMeta.size());
        get(Fields.Out, "task_count").setValue(out_row, 30L);
        putRow(data.outputRowMeta, out_row);
    }
    return true;
}
```

Figure 4.7: Pentaho DI — Custom Java Method.

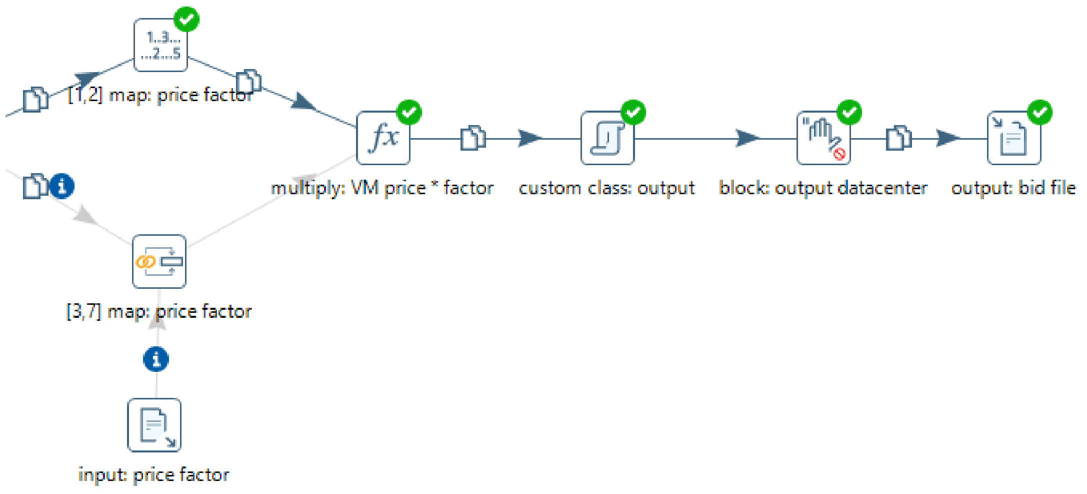


Figure 4.8: Pentaho DI — Prices and Output.

- The **execution time** is the period from the bid being submitted to the system and the allocation returns. It is also measured individually. This metric is focused on users that are only interested in the execution output (e.g. VM used on mathematical calculations where only the output is needed). We used 1-hour charging periods, so this metric is the allocation time plus 3600 seconds.

4.3.2 Efficiency

- **CPU Utilisation** Allows us to assess if the auction is increasing the number of allocations. A small value of this metric will lead to a low resource utilisation and consequently to low **Energy Efficiency** as more machines will be on idle state [44]. CPU utilisation is measured every five minutes;

- **Number of allocations** Divided into four categories: `demanding`, `restricted`, `relaxed`, and `conversion`. The sum of the categories is the total number of allocations.

- **Providers rating order** This metric allows us to measure the benefit of the reputation system. The order should change dynamically in each round when the provider quality changes.

4.3.3 Implementation

To gather and record the metrics, we implemented a ”Stats” class. The UML Class Diagram in 4.9 details its private attributes and the public API.

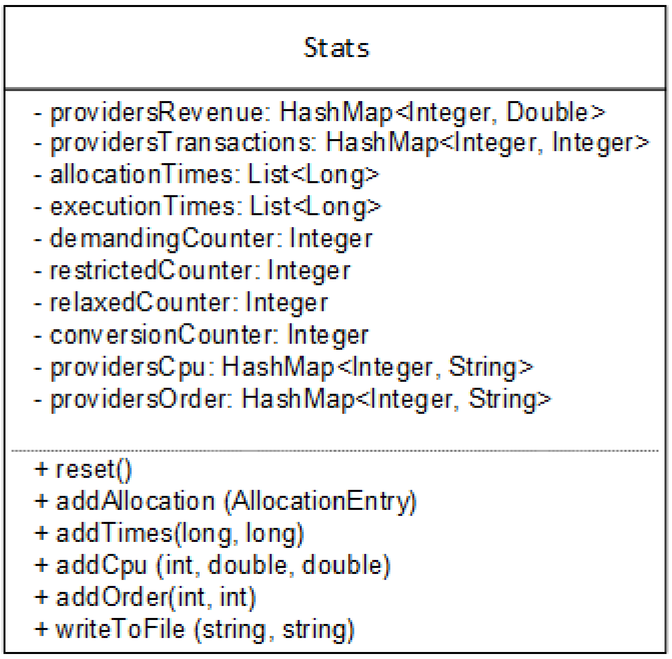

Stats
- providersRevenue: HashMap<Integer, Double>
- providersTransactions: HashMap<Integer, Integer>
- allocationTimes: List<Long>
- executionTimes: List<Long>
- demandingCounter: Integer
- restrictedCounter: Integer
- relaxedCounter: Integer
- conversionCounter: Integer
- providersCpu: HashMap<Integer, String>
- providersOrder: HashMap<Integer, String>
+ reset()
+ addAllocation (AllocationEntry)
+ addTimes(long, long)
+ addCpu (int, double, double)
+ addOrder(int, int)
+ writeToFile (string, string)

Figure 4.9: UML Class Diagram — Stats.

The first method resets the counters. It is called at the beginning of each dataset test. The ”addAllocation” method is called by ”AllocationManager” when the allocation is committed. It updates the following attributes: ”providersRevenue”, ”providersTransactions”, and the quality counters. The ”addTimes” is called when the cloudlet returns and has two parameters: allocation time and execution time. ”AddCpu” method receives the provider ID, its total used capacity and the total capacity. The attribute ”providersCpu” contains a history of the CPU utilisation. Therefore, this method appends to the corresponding string the new ratio $usedCapacity$ per $totalCapacity$. ”AddOrder” is invoked in every execution of the auction algorithm and is similar to ”addCpu”. It adds the new order to the corresponding string.

Finally, ”writeToFile” outputs the ”Stats” attributes to a text file, with the file name and location given by its parameters.

4.4 Benchmarks

We compare our system with one research work and one commercial system.

4.4.1 CloudAuctions

CloudAuctions is an auction system that does not consider QoS. It is described in [16], and the source code is available. To provide a fair comparison and compatibility with our bid representation language, CloudAuctions was improved in two ways: instead of the provider bid containing the VMs and their amount, it now contains only the total available resources — the algorithm will divide them to maximise the allocations. The second improvement is that the resources are restored once their allocation returns, and auction keeps repeating — a call market turned into a continuous market.

The allocation occurs if the user bid average price is higher than the provider bid average price, and the final price is also the average between the user and the provider average. User and provider bids were calculated with the Formula 4.1. Due to the first improvement, CloudAuctions will now use the Formula 4.2 to calculate the provider average bid price. This adapted formula does not affect the provider's profit: it increases the number of allocations but decreases the final price slightly. The limit K represents the number of different VM types in the bid.

$$AverageBidPrice(Bid) = \frac{\sum_{k=1}^{K} Bid.VMType[k].Amount \cdot Bid.VMType[k].UnitPrice}{\sum_{k=1}^{K} Bid.VMType[k].Amount} \tag{4.1}$$

$$AverageBidPriceAdapted(ProviderBid) = Bid.VMType["nano"].UnitPrice \tag{4.2}$$

4.4.2 AWS EC2 On-demand

We implemented a system similar to AWS EC2 On-demand. Contrary to FairCloud and CloudAuctions, the user bids are only sorted by their time-stamp. The matching is possible if the user price is higher than the provider price. The assignments do not consider degradations/conversions — the assigned amount is obtained by dividing the total capacity by each requested VM capacity.

This system is not an auction: the final price is always the provider price. This price was recorded in the same conditions as the user's bid price in the datasets: 25/06/2017, predefined 1h duration and Europe (Ireland). The provider prices are 0.111, 0.222, 0.444, 0.888 and 2.22 for the VMs `nano`, `micro`, `small`, `medium`, and `large`, respectively.

The prices are fixed and available on [45].

4.5 Results

In this section, we explore the allocation and execution time, allocations profile, CPU utilisation, average price, and the reputation order.

4.5.1 Allocation and Execution Time

The first test goal is to measure the allocation and execution time improvement. Graphic 4.10 represents the average allocation times for all algorithms and datasets.

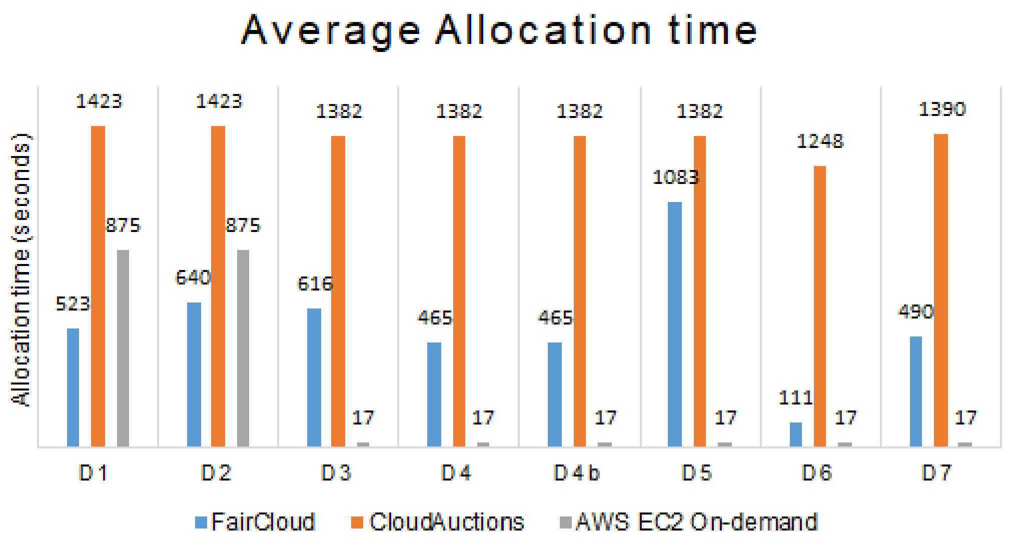


Figure 4.10: Graphic representing the allocation time.

The first remarkable difference is that FairCloud is faster than CloudAuctions in all datasets and

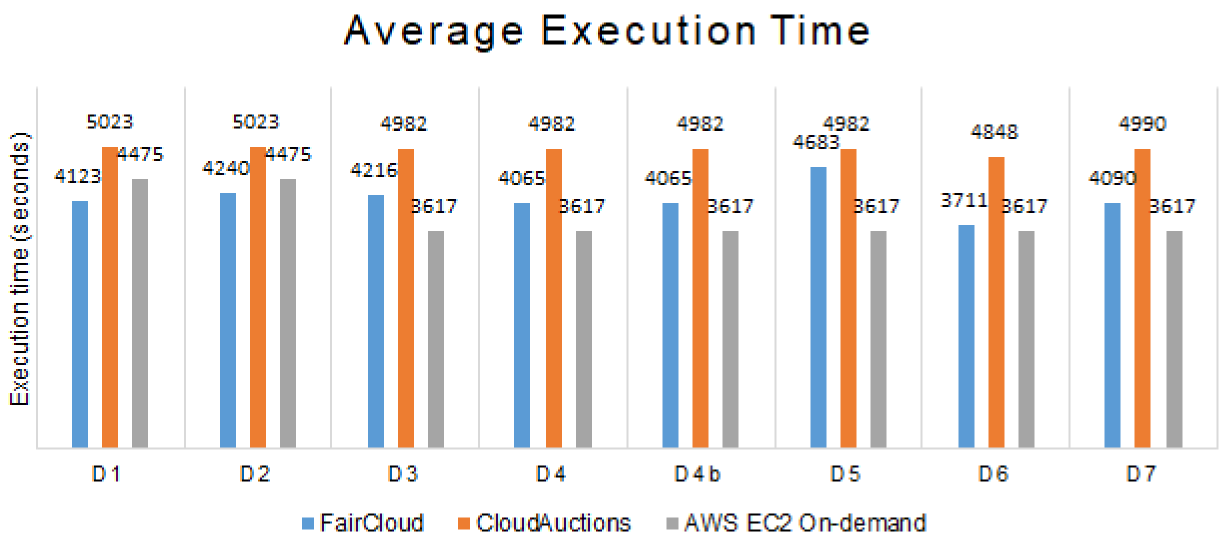


Figure 4.11: Graphic representing the execution time.

faster than AWS in Datasets 1 and 2. This time decrease is due to FairCloud allowing degradations in the requests. Later, in Section 4.5.2, we will detail the type and the amount of these degradations.

After Dataset 3, with heterogeneous prices, AWS is the fastest algorithm allocating requests but with a drawback. The majority of the requests are rejected. In Section 4.5.3 and 4.5.4, we analyse the effect on the CPU utilisation and the price, respectively.

Now, focusing on the performance of FairCloud in the different datasets. In Dataset 2, the performance decreased because by introducing different profiles some users do not allow degradations. FairCloud increased the speed in Dataset 3 because, with the range of prices, some requests are rejected due to their low prices, and consequently, there is more available capacity. Dataset 4 introduced one of our features, conversions, and it decreased the allocation time by 25%. It is important to notice that the reputation mechanism does not affect allocation time (Dataset 4b).

The highest allocation time is observed in Dataset 5. This is caused by the users requesting a minimum reputation score — the user bids need to wait for a provider with the desired reputation to be available. On the other hand, the allocation time is the lowest in Dataset 6. With a single high capacity provider, the FairCloud heuristic is optimal. Finally, the heterogeneous providers increase the fragmentation, slightly reducing the heuristic's effectiveness.

Figure 4.11 represents the average execution time. Although these values are higher, the relations between algorithms and datasets are the same as the allocation time.

4.5.2 Allocations Profile

The goal of this test is to measure the algorithms with a higher number of allocations and visualise the distribution of the different degradation profiles. Graphic 4.12 shows this information regarding Datasets 2, 3, 4, and 5 because they represent particular aspects: degradation profiles, pricing, conversions, and minimum reputation required, respectively.

We observe that FairCloud and CloudAuctions do the same number of allocations. However, the allocations on FairCloud were degraded, so they were executed in less time. Comparing Dataset 4 and 5 graphics (bottom), we conclude that the minimum reputation required increased the quality of the allocations: more 8188 allocations offer 100% resources and more 5375 offer 80%.

Finally, we see that AWS EC2 On-demand only allocates around 56% of the requests due to their pricing model. This static approach leads to resource waste.

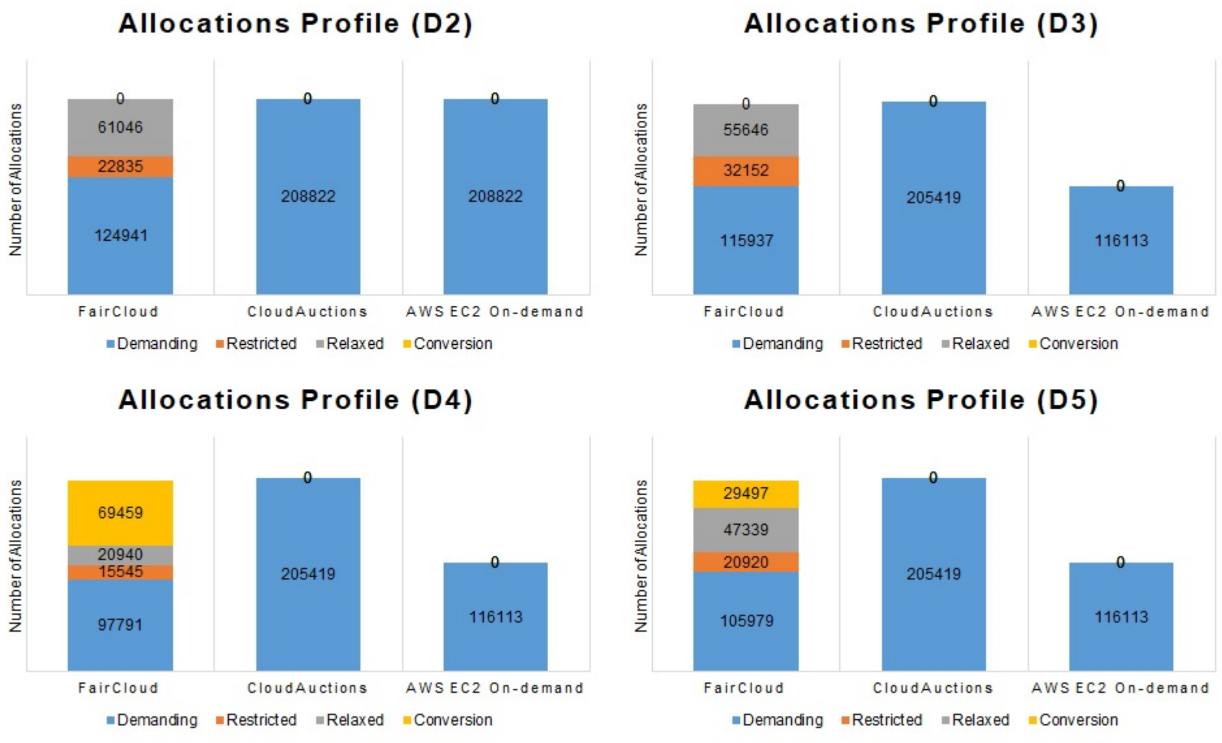


Figure 4.12: Graphic representing the number of allocations by profile.

4.5.3 CPU Utilisation

Graphic 4.13 represents for each time instant, the overall CPU utilisation. On the left, we observe that CloudAuctions and AWS EC2 On-demand have very close values. FairCloud has slightly less utilisation (46.7% vs 53.9%). This is because the requests are compacted and allocated sooner.

On the other hand, in Dataset 3, AWS overall utilisation is the lowest (29.8%), followed by FairCloud and AWS.

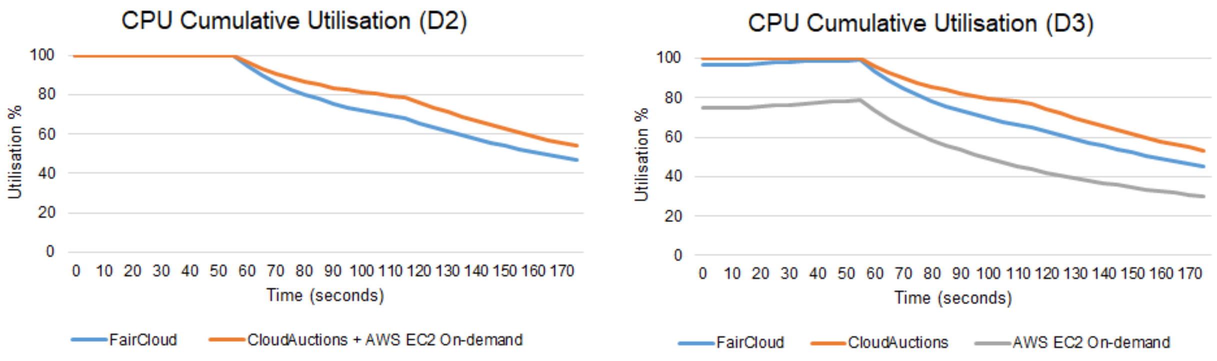


Figure 4.13: Graphic representing the CPU cumulative utilisation.

4.5.4 Average Price

Finally, in Graphic 4.14, we observe that the FairCloud average price is slightly lower than CloudAuctions because of two reasons. Our pricing model is more precise — we consider the unit price for each VM type and not the bid average price. Additionally, we offer a discount for users accepting degradations.

As expected, the AWS EC2 On-demand average price is the highest because the bids with the price below a limit are discarded.

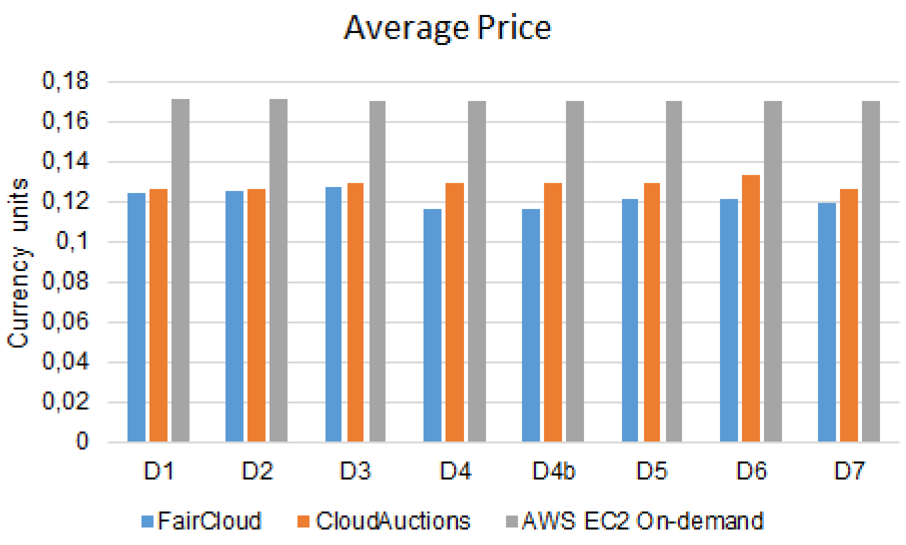


Figure 4.14: Graphic representing the average price per VM.

4.5.5 Reputation Order

Our algorithm implements a reputation system. With this test, we will see the benefit of using a hybrid ordering approach (price and reputation). Figure 4.15 represents the providers order (ranking) over time. Each colour represents a provider. In the first round (Time=0), the providers were ordered by price only (i.e., all had the maximum reputation). In the following rounds, the order changes: the providers with lower quality are now in the last places. As the time passes, we still see some minor adjustments. After the minute 90, the order stabilises.

We demonstrated that our reputation mechanism could detect providers with poor quality and benefit those with a slightly higher price but better QoS. On the other hand, the providers on AWS and CloudAuctions have a static order.

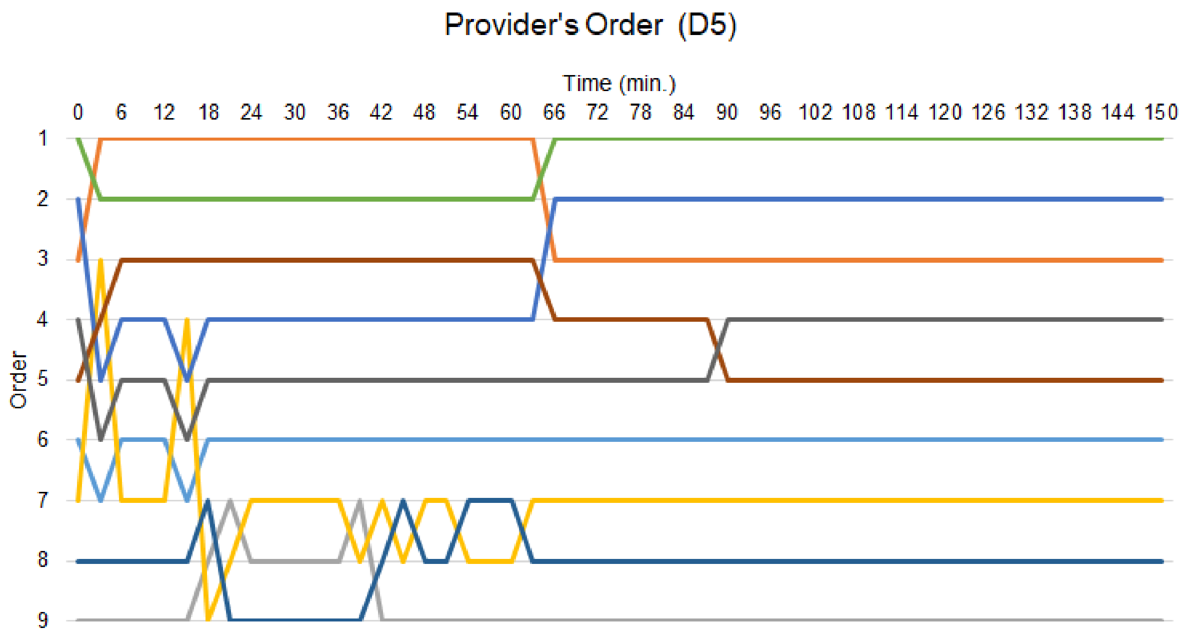


Figure 4.15: Graphic showing the Providers order in each algorithm execution.

4.6 Benefit to participants

After analysing the results, it is important to compare them and see what kind of participants benefit from using FairCloud. Starting with the users:

- Low/medium-end — by accepting to receive partial service, they get a discount and less execution time;
- High-end — they should specify in the bid that their profile is `demanding` to receive full service.

They will have maximum priority (i.e. user bids are sorted descending by price) and will, shortly, receive their resources.

On the other hand, FairCloud cannot account for ultra-high users (i.e., users with unlimited budget and very demanding requirements). They can opt for a traditional system, like AWS EC2 On-demand, for a swift allocation time and a predefined price.

Similarly, cloud providers have advantages in bidding in FairCloud:

- They receive more requests, increase the CPU utilisation and, consequently, their energy efficiency;
- The reputation mechanism assures that the providers with better quality receive the most requests;
- The price per VM will be lower, but the overall profit will increase;
- They still have control over the price. FairCloud pricing formula considers both the user's and provider's price.

5

Conclusion

Contents

We started this work by exploring the energy consumption in cloud computing and ways of optimising it. One of the ways is to reduce the idle run of the system, increasing utilisation (i.e., increased energy consumption but producing relatively more output). We further studied this concept by analysing two examples: a study of CPU utilisation in more than 5,000 machines and the workstation occupation in a university cluster. We observed that they are in sub-utilisation. Finally, we gave two examples of CPU and datacenter optimisation. The introduction ends by presenting the different cloud pricing models: pay-as-you-go, subscription, pay-for-resources and, the latest, dynamic.

Our proposed objectives were to formulate a taxonomy, and classify research and commercial systems according to it. Then, we proposed to implement and evaluate a system respecting the desired properties (i.e., reduce the idle time, reduce allocation time, and use a dynamic pricing model).

Regarding the taxonomy, we identified the core categories: resource type, participants, bidding scheme, reservation, market, bid representation language, and the goals. We also studied a scheduler classification: static vs dynamic, and local vs global. Followed by the classification of algorithms and the detail of some disruptive features.

We started the architecture by presenting the desirable properties for a cloud-auction: continuous, truthful, combinatorial and facilitate the resource allocations. FairCloud considers QoS requirements, in particular, degradations, conversions, heuristics to maximise allocations and their quality, and a reputation mechanism. The core algorithms are the bid sorting and matching, update heuristics and auction engine. They were complemented by data structures, the assignment protocol, roll-back and the reputation operations. To wrap-up this section, FairCloud was assessed considering our taxonomy.

We simulated FairCloud in CloudSim, using one of its extensions — CloudAuctions. The bid representation language was implemented in the classes "Bid", "DatacenterBrokerBid", and "DatacenterBid". Our protocol is supported by entities: "SimEntity", "CoreAuctioneer", "AllocationManager", "DatacenterBroker", "BidderDatacenterBroker", "Datacenter", and "BidderDatacenter".

Finally, we evaluated our system using the Google Cluster Data, which represents real user requests to Google Cluster. We described and implemented in Pentaho DI eight datasets, with multiple configurations (e.g. price, number of providers, user's degradation profile). Our metrics were the average price per VM, allocation and execution time, CPU utilisation, number of allocations, and providers rating order. We compared FairCloud with a research system, CloudAuctions, and a commercial static price system, AWS EC2 On-demand.

We observed that we provide the fastest allocation time in the datasets with a fixed price. This happened because we were able to degrade allocations, but always according to the user profile. In datasets with variable price, FairCloud was faster than CloudAuctions but slower than AWS EC2 because the requests were rejected. We could study this result further by comparing the CPU utilisation. Finally, we also demonstrated that our reputation algorithm could order the providers based on their quality.

We concluded that our system benefits low/medium-end users because, by accepting a partial service, they get a discount and less execution time. High-end users can specify that their profile is `demanding` to receive full service. They will have maximum priority and will, shortly, receive their services. On the other hand, ultra-high-end users can opt for a traditional system, like AWS EC2 On-demand. Providers also benefit from using FairCloud as they will receive more requests, increasing their profit and energy efficiency. They also maintain partial control over the price.

5.1 Future Work

Some suggestions for improvements are: FairCloud could allow the participants to specify their geographic preferences (e.g. location that increases redundancy or close provider for less latency). Also, a history of past prices would decrease the rejected requests rate. To enrich the pricing model, we could implement subscription and a variable request duration. This way, the users would still have resources if their auction bid did not succeed.

We could improve the evaluation by adding a third benchmark — AWS EC2 Spot Instances, and measuring the energy consumption, using the CloudSim power consumption models.

Bibliography

[1] M. Dayarathna, Y. Wen, and R. Fan, “Data Center Energy Consumption Modeling: A Survey,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.

[2] L. A. Barroso and U. Hölzle, “The Case for Energy-Proportional Computing,” *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[3] RNL, “Statistics,” (Accessed: 01/09/2017). [Online]. Available: https://rnl.tecnico.ulisboa.pt/estatisticas

[4] H. Zhang, H. Jiang, B. Li, F. Liu, A. V. Vasilakos, and J. Liu, “A framework for truthful online auctions in cloud computing with heterogeneous user demands,” *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 805–818, 2016.

[5] D. J. Dubois and G. Casale, “OptiSpot: minimizing application deployment cost using spot cloud resources,” *Cluster Computing*, vol. 19, no. 2, pp. 893–909, 2016.

[6] S. Son and K. M. Sim, “A Price- and-Time-Slot-Negotiation Mechanism for Cloud Service Reservations,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 3, pp. 713–728, 2012.

[7] P. Mell and T. Grance, “The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology,” *National Institute of Standards and Technology, Information Technology Laboratory*, vol. 145, p. 7, 2011.

[8] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, p. 50, 2010.

[9] J. Kaplan, W. Forrest, and N. Kindler, “Revolutionizing data center energy efficiency,” *McKinsey & Company, Tech. Rep*, 2008.

[10] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, “Cloud Computing,” *ACM Computing Surveys*, vol. 47, no. 2, pp. 1–36, 2014.

[11] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, “A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems,” *Advances in Computers*, vol. 82, pp. 47–111, 2010.

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google file system,” *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, p. 29, 2003.

[13] K. Li, “Improving Multicore Server Performance and Reducing Energy Consumption by Workload Dependent Dynamic Power Management,” *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 122–137, 2016.

[14] D. Lucanin and I. Brandic, “Pervasive Cloud Controller for Geotemporal Inputs,” *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 180–195, 2016.

[15] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, and I. Ahmad, “Cloud Computing Pricing Models: A Survey,” *International Journal of Grid and Distributed Computing*, vol. 6, no. 5, pp. 93–106, 2013.

[16] P. Samimi, Y. Teimouri, and M. Mukhtar, “A combinatorial double auction resource allocation model in cloud computing,” *Information Sciences*, vol. 357, pp. 201–216, 2016.

[17] Y. Zhao, Z. Huang, W. Liu, J. Peng, and Q. Zhang, “A combinatorial double auction based resource allocation mechanism with multiple rounds for geo-distributed data centers,” *2016 IEEE International Conference on Communications, ICC 2016*, 2016.

[18] W. Shi, L. Zhang, C. Wu, Z. Li, and F. C. Lau, “An online auction framework for dynamic resource provisioning in cloud computing,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, pp. 71–83, 2014.

[19] R. Zhou, Z. Li, C. Wu, and Z. Huang, “An Efficient Cloud Market Mechanism for Computing Jobs With Soft Deadlines,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 793–805, 2017.

[20] O. Agmon Ben-Yehuda, E. Posener, M. Ben-Yehuda, A. Schuster, and A. Mu’alem, “Ginseng: Market-Driven Memory Allocation,” *ACM SIGPLAN Notices*, vol. 49, no. 7, pp. 41–52, 2014.

[21] R. P. Mcafee and J. McMillan, “Auctions and Bidding,” *Journal of Economic Literature*, vol. 25, no. 2, pp. 699–738, 1987.

[22] S. Yi, D. Kondo, and A. Andrzejak, “Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud,” in *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE, 2010, pp. 236–243.

[23] Y. Zhang, K. Xu, X. Shi, H. Wang, J. Liu, and Y. Wang, “Continuous double auction for cloud market: Pricing and bidding analysis,” in *2016 IEEE Wireless Communications and Networking Conference*. IEEE, 2016, pp. 1–6.

[24] I. Fujiwara, K. Aida, and I. Ono, “Applying double-sided combinational auctions to resource allocation in cloud computing,” *Proceedings - 2010 10th Annual International Symposium on Applications and the Internet, SAINT 2010*, pp. 7–14, 2010.

[25] M. V. Dijk and A. Juels, “On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Computing,” *Proceedings of the 5th USENIX Conference on Hot Topics in Security*, pp. 1–8, 2010.

[26] D. Chen and H. Zhao, “Data Security and Privacy Protection Issues in Cloud Computing,” *2012 International Conference on Computer Science and Electronics Engineering*, no. 973, pp. 647–651, 2012.

[27] Z. Chen, L. Chen, L. Huang, and H. Zhong, “On Privacy-Preserving Cloud Auction,” in *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2016, pp. 279–288.

[28] T. Casavant and J. Kuhl, “A taxonomy of scheduling in general-purpose distributed computing systems,” *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141–154, 1988.

[29] J. Simao and L. Veiga, “Partial Utility-driven Scheduling for Flexible SLA and Pricing Arbitration in Clouds,” *IEEE Transactions on Cloud Computing*, no. 99, pp. 1–1, 2014.

[30] C. Qu, R. N. Calheiros, and R. Buyya, “A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances,” *Journal of Network and Computer Applications*, vol. 65, pp. 167–180, 2016.

[31] Google Cloud, “Google Preemptible VMs,” (Accessed: 01/09/2017). [Online]. Available: https://cloud.google.com/preemptible-vms/

[32] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, “Deconstructing Amazon EC2 Spot Instance Pricing,” in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, vol. 1, no. 3. IEEE, 2011, pp. 304–311.

[33] J. Simao and L. Veiga, “Flexible SLAs in the Cloud with a Partial Utility-Driven Scheduling Architecture,” in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 1. IEEE, 2013, pp. 274–281.

[34] Y. H. Wang and I. C. Wu, “CloudSim: a toolkit formodeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software - Practice and Experience*, vol. 39, no. 7, pp. 701–736, 2009.

[35] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, “ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers,” *Software: Practice and Experience*, pp. 505–521, 2016.

[36] P. Kathiravelu and L. Veiga, “Concurrent and Distributed CloudSim Simulations,” in *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*. IEEE, 2014, pp. 490–493.

[37] C. R. Hellerstein, J. Wilkes, and J. L., “{Google} cluster-usage traces: format + schema,” Google Inc., Mountain View, CA, USA, Tech. Rep., 2011.

[38] John Wilkes, “More {Google} cluster data,” *Google research blog*, 2011.

[39] N. El-Sayed, H. Zhu, and B. Schroeder, “Learning from Failure Across Multiple Clusters: A Trace-Driven Approach to Understanding, Predicting, and Mitigating Job Terminations,” *Proceedings - International Conference on Distributed Computing Systems*, pp. 1333–1344, 2017.

[40] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. a. Kozuch, “Heterogeneity and dynamicity of clouds at scale : Google Trace Analysis,” *Proceedings of the Third ACM Symposium on Cloud Computing - SoCC ’12*, pp. 1–13, 2012.

[41] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, “Towards characterizing cloud backend workloads,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, p. 34, 2010.

[42] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. M. Lau, “Online Auctions in IaaS Clouds: Welfare and Profit Maximization With Server Costs,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1034–1047, 2017.

[43] AWS, “Amazon EC2 Spot Instances Pricing,” (Accessed: 01/09/2017). [Online]. Available: https://aws.amazon.com/ec2/spot/pricing/

[44] L. Sharifi, N. Rameshan, F. Freitag, and L. Veiga, “Energy Efficiency Dilemma: P2P-cloud vs. Datacenter,” in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. IEEE, 2014, pp. 611–619.

[45] AWS, “Amazon EC2 Pricing,” (Accessed: 01/09/2017). [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/