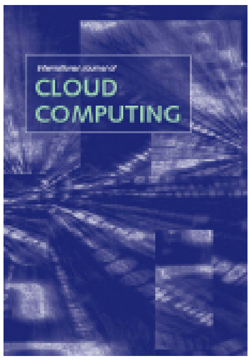


International Journal of Cloud Computing

ISSN online: 2043-9997 - ISSN print: 2043-9989
https://www.inderscience.com/ijcc

A distributed auction-based algorithm for virtual machine placement in multiplayer cloud gaming infrastructures

Yassine Boujelben, Hasna Fourati

DOI: 10.1504/IJCC.2024.10048138

Article History:

Received:	11 April 2022
Accepted:	21 April 2022
Published online:	26 January 2024

A distributed auction-based algorithm for virtual machine placement in multiplayer cloud gaming infrastructures

Yassine Boujelben*

ENETCOM,
University of Sfax,
Sfax, Tunisia
Email: yassine.boujelben@enetcom.usf.tn
*Corresponding author

Hasna Fourati

Digital Research Centre of Sfax,
University of Sfax,
Sfax, Tunisia
Email: hasna.fourati21@gmail.com

Abstract: Cloud gaming is an emerging service model that basically mimics the cloud computing model. Indeed, intensive computing tasks incurred by the graphical processing of the fairly complex game scenes are exported to remote cloud servers. While this would alleviate the hardware and software requirements on the gaming terminals, it poses serious problems of quality of service and experience. Furthermore, as the massive multiplayer gaming model becomes increasingly popular, computing resources are likely spread across multiple data centres and the need for a distributed assignment algorithm becomes paramount. In this paper, we are interested in the assignment of virtual machines hosted on rendering servers in a distributed cloud gaming infrastructure to requests sent by online gamers. We use the auction algorithm along with several efficient extensions to solve the virtual machine placement problem. We propose a completely distributed implementation technique without any shared memory for our algorithm called DVMP.

Keywords: multiplayer cloud gaming; MCG; virtual machine placement; VMP; matchmaking; distributed auction algorithm; distributed VMP; gaming experience.

Reference to this paper should be made as follows: Boujelben, Y. and Fourati, H. (2024) ‘A distributed auction-based algorithm for virtual machine placement in multiplayer cloud gaming infrastructures’, *Int. J. Cloud Computing*, Vol. 13, No. 1, pp.80–98.

Biographical notes: Yassine Boujelben received his Dipl-Ing in Telecommunications from the École Supérieure des Communications de Tunis, Tunisia, in 1996, MSc in Systems Analysis and Digital Processing from the École Nationale d’Ingénieurs de Tunis, Tunisia, in 1996, and

PhD in Telecommunications from the Institut National de la Recherche Scientifique, Centre Énergie Matériaux Télécommunications, University of Quebec, Canada, in 2003. He is an Associate Professor with the École Nationale d'Électronique et des Télécommunications de Sfax, University of Sfax, Tunisia. He had two postdoctoral research positions with the École Polytechnique de Montréal, Canada, and the Group for Research in Decision Analysis (GERAD), Montreal. From 2006 to 2009, he joined the Tektronix Communications, where he served as a system design engineer. His current research interests include network support for cloud gaming, QoS management for multimedia broadcast/multicast services in 5G/6G, and resource optimisation in IoT networks.

Hasna Fourati received his Master's in Telecommunication and Network Systems from the University of Sfax in 2017. She is a researcher at the Laboratory of Signals, Systems, Artificial Intelligence and Networks, Digital Research Center of SFAX. Her research interests are 5G, artificial intelligence and cloud computing.

1 Introduction

Modern computer online games are often graphically intensive. Softwares supporting these games are becoming more and more complex and they quickly become incompatible with low performance computers like tablets and smartphones. Therefore, trying a new game leads to consuming more time, which imposes heavy burdens on users and requires them to reconfigure or upgrade their computers. In recent years, a new type of cloud service called '*cloud gaming*' has emerged and is becoming more and more popular in the gaming industry. Cloud gaming is a promising solution to reduce loads on terminals while minimising maintenance costs. Indeed, cloud gaming relies on the use of cloud computing technologies to build large-scale gaming infrastructures in order to improve responsiveness and scalability, and overcome the hardware constraints of the light clients. The benefits of cloud computing that online gaming could take advantage of include ubiquity, multi-platform to provide immersive gaming space, reduced power consumption and the presence of multiple cloud services, such as storage (D'Angelo et al., 2015; Cai et al., 2014; Soliman et al., 2013). Therefore, cloud gaming might be viewed as a service providing video games on demand to consumers through the use of cloud technologies.

A cloud gaming system must collect the actions from players, transmit them to the cloud server, process the action, render the results, code or compress the resulting changes, and retransmit the scenes as a video sequence from the rendering server back to the player (Shea et al., 2013). Consequently, for terminals which computing capacity does not allow them to high quality games, game logics and intensive graphical processing are executed in the cloud and the scenes are returned as video clips through the internet ready to be displayed. This requires very tight interaction time limits between the terminal and the servers. Actually, it has been shown in Jarschel et al. (2011) through subjective measurements that a delay of 80 ms is a threshold for good quality perception scores in the case of slow and medium paced games such as sports and role playing games. However, for fast paced games like first person shooter (FPS), this delay threshold does only provide medium quality perception scores because of the problems of consistency and violation of causal order that long delays could induce, especially for multiplayer games as studied by Boujelben et al. (2006) and Dammak et al. (2018).

Choy et al. (2012) conducted a set of network delay measurements to test if the latency threshold of 80 ms can be achieved on existing commercial cloud computing infrastructures. They found that if the Amazon EC2 cloud with its three data centres (DCs) was used to support cloud gaming in the USA, only 70% of the population will enjoy a good perceived quality as their network delay will be under the 80 ms threshold. They also found that 10% of the population will be considered unreachable as their network delay will exceed 160 ms.

In order to address this problem, some potential solutions were suggested. The most trivial one was to increase the number of DCs. However, in addition to not being economically viable, this solution could not even completely solve the problem as shown in Choy et al. (2012) with 20 DCs. The second solution is to support the existing cloud computing infrastructure with additional resources. Choy et al. (2014) proposed the use of edge servers such as those deployed by content delivery networks (CDN) in order to be closer to the gamers, which consequently reduces the latency. In a similar approach, Lin and Shen (2017) suggested the deployment of an intermediate layer between the cloud and the gamers called the *fog*, where supernodes are placed to perform graphical rendering. The third solution consists in optimising either or both the virtual machine (VM) placement in the cloud and the routing of requests.

In this paper, we will be interested in the virtual machine placement (VMP) problem which is considered the cornerstone of several cloud service optimisation and deployment problems (Laghrissi and Taleb, 2019). The cloud servers are distributed geographically and when a player attempts to connect to the system and starts playing, a VMP problem arises. Indeed, several constraints relating to resource availability and quality of experience must be considered when selecting the optimal VM, which makes this problem rather complex. The client must choose the closest rendering server in terms of network delay to improve interactivity and therefore the user experience. However, this server may already be too busy, or it may not have sufficient resources to handle games with high graphic requirements, or it may not contain the requested game since the game provider offers hundreds of games and there is no need to put all of them on every server. Moreover, if the customer chooses a multiplayer game, one should consider the location of the other players participating in the game, a process commonly called *matchmaking* (Guan et al., 2017).

The objective of our work is to propose a distributed VMP algorithm in the context of online games in a distributed cloud infrastructure. Indeed, unlike most previous works which assume the existence of a single central server responsible for the execution of the VMP algorithm (Chaisiri et al., 2009; Alicherry and Lakshman, 2012; Hao et al., 2017; Saxena et al., 2021), we will consider in this work a network of servers which run a distributed algorithm. This will solve the problem of scale inherent to massively multiplayer online games (MMOG). Our proposed solution is based on the auction algorithm that solves the assignment problem according to a highly parallelised procedure which will later make it possible to build a decentralised assignment procedure in order to choose the optimal VM for each request. The major contributions of our work are as follows:

- We introduce in the distributed cloud gaming architectures a new decision layer in between the players and the DCs that hosts the rendering servers. This layer is formed of an overlay network of access devices, e.g., portal servers, interconnected with internet links. These devices will run the distributed VMP algorithm.
- We formulate the VMP problem in this new three layer architecture as a transshipment problem. Then we transform it first to a transportation problem and finally to an assignment problem emphasising the game traffic path.
- We design a completely distributed auction-based VMP algorithm along with multiple extensions from existing literature. Through extensive computations, we show the effectiveness of our proposition in terms of computation time and optimality gap.

The rest of this paper is organised as follows. In Section 2, we present some related works. In Section 3, we describe the VMP problem and propose a relevant formulation. In Section 4, we briefly describe the native auction algorithm, then we discuss some extensions to make it more efficient and more adapted to our VM assignment scenario, and finally we describe our distributed virtual machine placement (DVMP) algorithm. In Section 5, we present the DVMP performance results and conclude in Section 6.

2 Related works

The VMP problem has been extensively studied in the last decade as can be ascertained in these surveys by Masdari et al. (2016), Zhang et al. (2016) and Laghrissi and Taleb (2019), and it continues to generate increasing research interest with the evolution of cloud architectures and services. Previous works can be divided according to the cloud architecture. Thus, some works have considered optimisations in a single DC, and others in distributed cloud infrastructures. Finally, a specific class can also be reserved to the multiplayer cloud gaming (MCG) infrastructures.

Within a single DC, Meng et al. (2010) approached the VMP problem from two perspectives. First, they considered minimising the aggregate traffic rates on every switch given a traffic matrix, and second, they used a clustering technique to solve the scalability problem. Saxena et al. (2021) considered instead energy consumption and security as main objectives. Finally, Peake et al. (2022) focused on the algorithmic aspects of the problem by proposing a parallel solving technique based on the ant colony optimisation. Overall, algorithmic aspects dominated previous work in this category with an awareness of the scalability constraint.

Handling the VMP problem in multi-DC architectures considered mainly the multi-cloud or the distributed cloud models. Chaisiri et al. (2009) proposed an optimal solution for the VMP problem in a multiple cloud provider environment. They introduced a central unit called *cloud broker* which is responsible for solving the placement problem. In this same brokering context, Kessaci et al. (2013) proposed a metaheuristic searching method based on the genetic algorithm. Alicherry and Lakshman (2012) were interested in optimising the network performance in distributed cloud environments. They proposed a two-phase solution approach based on choosing first the optimal DC, then optimising the placement inside this DC. They also proposed a clustering technique of the resource requests in order to minimise the inter-DC traffic. Hao et al. (2017) proposed an online methodology to allocate VMs to geographically distributed cloud locations. Deciding to accept or refuse a request is made separately for each request without looking at future arrivals. Parida and Pati (2020) used the cloud brokering policy in a distributed cloud.

Hong et al. (2015) proposed a cloud gaming architecture based on a broker which is responsible for both VM placement and traffic and workload monitoring. Deng et al. (2018) considered the VMP problem in the context of MCG. They discussed typical architectures where rendering servers are placed between the players and the main game server. A matchmaking process ensures that every player is served by a rendering server which may serve more than one player. Zhang et al. (2019) extended the server rendering placement to the context of virtual reality in a distributed mobile edge cloudlet.

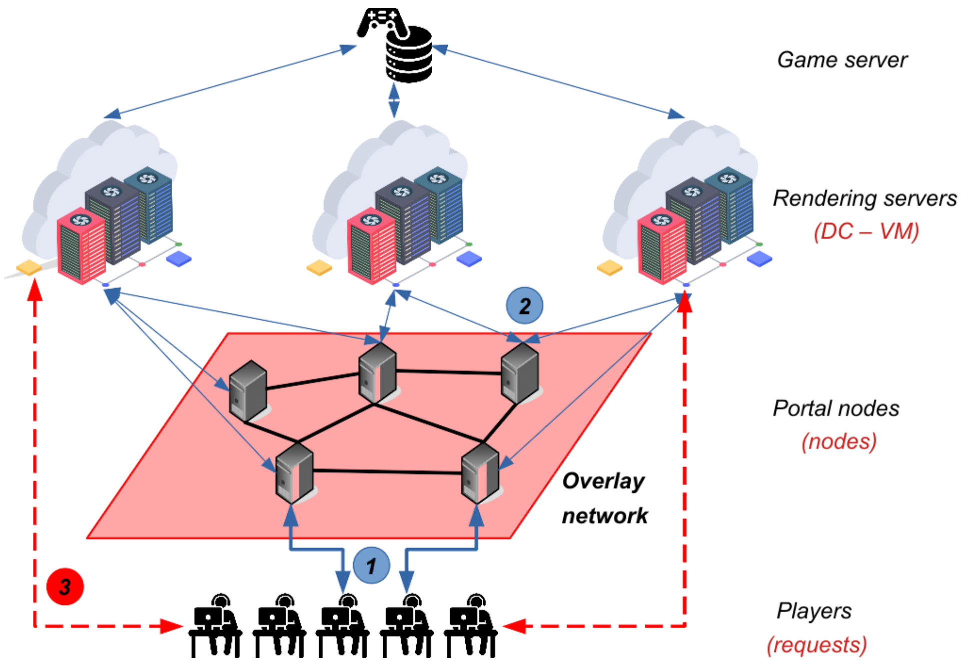
Despite the diversity of cloud architectures considered in previous works, they are mainly based on a central entity responsible for the VMP problem solving, like the cloud broker in most proposals. The only work we are aware of which considered a distributed algorithm for the VMP problem is that of Han et al. (2020) who proposed a distributed implementation of a VMP algorithm based on game theory. The players determines a set of beneficial servers and send it to a centralised dispatcher which makes the final allocation decision. This work is the closest to ours. However, instead of assigning the main task of executing the distributed algorithm to the client terminals, we will propose an auction-based completely distributed algorithm implemented on portal nodes which are owned by the game provider. Our algorithm will not need a centralised entity to make the final allocation decision.

Auction-based solutions have received particular interest in recent years to solve resource allocation problems and particularly the VMP problem. The most used auction variant in the context of cloud computing is the combinatorial auction (CA) where bidders can place bids on combinations of items, e.g., CPU, memory, storage and networking resources. Zaman and Grosu (2013) proposed a centralised greedy version of the CA for a single cloud provider, where a central entity collects bids from the users then computes the allocation. Samimi et al. (2016) proposed also a centralised version of the CA based on an *auctioneer* which runs the resource allocation algorithm. Li et al. (2018), Tan et al. (2020) and Zhang et al. (2020) were interested in an online implementation of the auction. As we can see through all these implementations, our proposal for a distributed implementation of an auction-based VMP solution has not been addressed in previous work.

3 Problem description and formulation

We will use the basic architecture of a cloud-based online gaming infrastructure as shown in Figure 1. We assume that a game as a service (GaaS) provider advertises, through its website, a set of games to its customers located all over the world. We also assume that a set of *portal nodes* are deployed in appropriate access zones in order to bring the service closer to the customers. These nodes are interconnected in an overlay network and will run de VMP distributed algorithm. When a customer, a player in our case, wants to use this service, it has to connect first to a portal node in order to choose the game and to be authenticated through a portal server. A server selection mechanism, e.g., based on local DNS server location, should route the customer's request to the closest portal node. Second, the portal server has to select an *appropriate* rendering server for each request to provide a VM. Rendering servers are responsible for providing the necessary computing, graphical processing and storage resources for the smooth running of the game. They are hosted in geographically distributed DCs and connected to the game server. Finally, once a VM is assigned to a request, the corresponding player could connect directly to the hosting server on a specific DC to start playing.

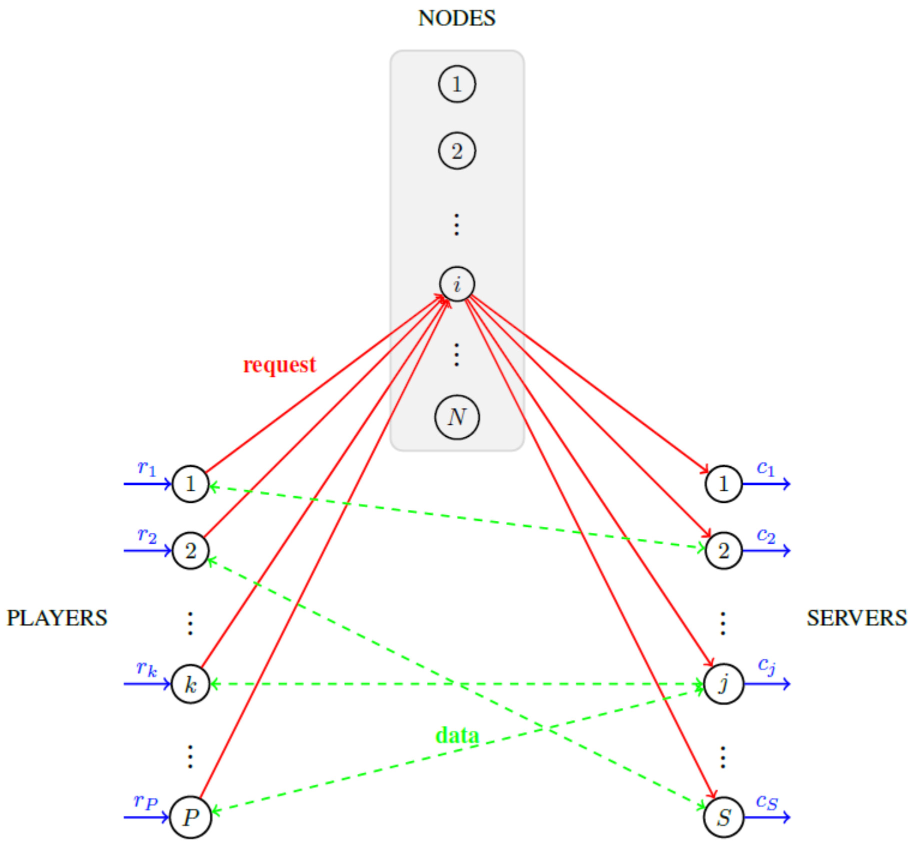
Figure 1 Cloud gaming architecture (see online version for colours)



Our approach describes this two-stage selection process as a transshipment problem where a set of requests (*supply*) have to be sent to a set of rendering servers (*demand*) via a set of intermediate nodes as shown in Figure 2. We assume that at the moment of matchmaking, there is a set of P players each one of them sends a request r_k, $k \in P$, identified by CPU and memory requirements to N portal nodes. We also assume that S rendering servers are available. Each server j is identified by its capacity c_j in CPU and memory resources.

Nevertheless, our problem could have some particularities. Actually, as we can see in the network diagram of Figure 2, the request paths are different from the traffic paths. Therefore, if the goal is to optimise the connection establishment process between a player and a rendering server, including the selection of the best server in terms of resource availability and the speed of configuring a VM, then we have to solve the transshipment problem with the requests as the shipped commodities. However, if the goal is to optimise the quality of experience of the player throughout the game, then we have to consider some performance metrics on the end-to-end paths between players and servers.

Figure 2 Network diagram for the transshipment problem (see online version for colours)



In this work, we will only consider the first case where a set of requests have to be routed to the rendering servers via the intermediate nodes. Since it is not the request's route that will be the most important for the player but rather to be assigned to the best server that could offer him the best service, we will simplify the problem by assuming that every subset of players is statically assigned to a particular intermediate node, based for example on geographical proximity or any other relevant metric. We will also express the resources in terms of standard units of VMs. That means that a VM will be assigned for one request and the server capacity c_j will be expressed in number of VMs as well. Therefore, the transshipment problem could be reduced to a transportation problem between nodes and rendering servers as shown in Figure 3. Each node i will have a number of requests n_i for VM configurations across the S servers.

Let a_{ij} be the profit of serving a VM request from node i on server j, and x_{ij} be a variable indicating the number of VM requests from node i assigned to server j. The VMP problem could be formulated as follows

$$\max \quad \sum_{i=1}^{N} \sum_{j=1}^{S} a_{ij} x_{ij} \tag{1}$$

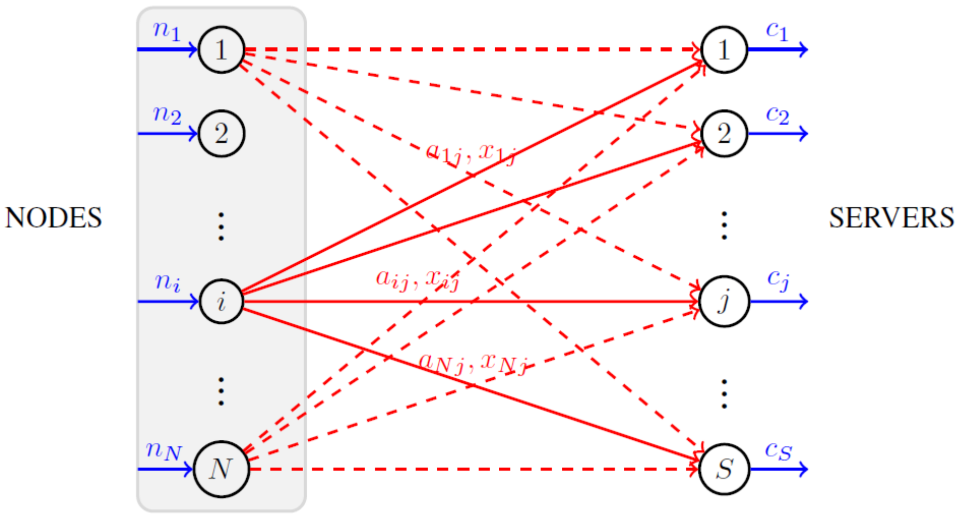
$$\sum_{j=1}^{S} x_{ij} \leq n_i \quad \forall i = 1, ..., N \tag{2}$$

$$\sum_{i=1}^{N} x_{ij} \leq c_j \quad \forall j = 1, ..., S \tag{3}$$

$$x_{ij} \geq 0 \text{ and integer} \quad \forall (i, j) \tag{4}$$

The problems (1)–(4) defines an unbalanced transportation problem. Indeed, at optimality, either set of constraints (2) or (3) will be satisfied as equality. The problem could be balanced by adding dummy nodes or servers and centrally solved using well-known algorithms. However, centralised approaches are generally not useful for real-time decision-making of large-scale problems. We will propose in this work an approach based on the auction algorithm which offers the opportunity for a decentralised implementation.

Figure 3 Description of the VM assignment as a transportation problem (see online version for colours)



4 Distributed virtual machine placement

In this section, we first describe how we could use a distributed version of the auction algorithm to solve our transportation problem. Then, we present an extension based on request bundling in order to reduce the problem size.

4.1 The auction algorithm

The auction algorithm is initially proposed to solve the assignment problem where N persons have to be assigned to N objects on a one-to-one basis so as to maximise a total profit (Bertsekas, 1992).

In order to convert problems (1)–(4) into an assignment problem, we duplicate each node as many times as it has requests, i.e., node i is duplicated n_i times. We do the same thing with the servers that will be duplicated according to their capacities in terms of VMs. Finally, we will have a problem of assigning $P = \sum_{i=1}^{N} n_i$ requests to the $C = \sum_{j=1}^{S} c_j$ VMs. We assume that the problem is balanced, that is $P = C$.

Let every VM v in server j, noted in the following v_j, has a price $p_{v_j} \geq 0$, that each request must pay to have it. We assume that the profit of assigning any request r in node i, noted in the following r_i, to any VM v_j will be $a_{r_i v_j}$. Basically, the assignment profit of any request from node i to any VM in server j will be the same as a_{ij}. However, we can assume that a node may need to add priorities or classes of service to requests allowing it to offer differentiated services to its customers (players). Therefore, we define an assignment $\mathcal{A}$ as the set of pairs (r_i, v_j).

The net value of VM v_j for any request r_i is $a_{r_i v_j} - p_{v_j}$. Every request r_i would like to be assigned to a VM v_j that provides it with a maximum net value:

$$a_{r_i v_j} - p_{v_j} = \max_{\substack{\{l=1,\ldots,S\} \\ \{u=1,\ldots,c_l\}}} \{a_{r_i u_l} - p_{u_l}\} \tag{5}$$

An equilibrium assignment is found when the net value for every request r_i assigned to VM v_j is within a constant $\epsilon > 0$ of being maximal, that is

$$a_{r_i v_j} - p_{v_j} \geq \max_{\substack{\{l=1,\ldots,S\} \\ \{u=1,\ldots,c_l\}}} \{a_{r_i u_l} - p_{u_l}\} - \epsilon \tag{6}$$

for all requests r_i.

The main iteration of the auction algorithm consists of two phases: the bidding and the assignment. In the following, we will briefly describe these two phases, more details could be found in Bertsekas (1992). We assume that an assignment $\mathcal{A}$ (may be empty) exists and we consider a subset $\mathcal{R}$ of unassigned requests.

- *Bidding phase:* For each request $r_i \in \mathcal{R}$ which has not be assigned under $\mathcal{A}$, find the best VM v_j^* such that:

  $$v_j^* = \arg \max_{\substack{\{l=1,\ldots,S\} \\ \{u=1,\ldots,c_l\}}} \{a_{r_i v_j} - p_{u_l}\} \tag{7}$$

  having the maximum net value:

  $$x_{r_i} = \max_{\substack{\{l=1,\ldots,S\} \\ \{u=1,\ldots,c_l\}}} \{a_{r_i u_l} - p_{u_l}\}, \tag{8}$$

  and the best net value offered by a VM other than v_j^*:

  $$y_{r_i} = \max_{\substack{\{l=1,\ldots,S\} \\ \{u=1,\ldots,c_l\} \\ u_l \neq v_j^*}} \{a_{r_i u_l} - p_{u_l}\}. \tag{9}$$

  Compute the *bid* of request r_i given by:

  $$b_{r_i v_j^*} = p_{v_j^*} + x_{r_i} - y_{r_i} + \epsilon. \tag{10}$$

- *Assignment phase:* Each VM v_j receives bids from a set of requests $\mathcal{P}(v_j)$ in the bidding phase, increases its price to the highest bid:

$$p_{v_j} := \max_{r_i \in \mathcal{P}(v_j)} b_{r_i v_j}, \tag{11}$$

  and gets assigned to the highest bidder r_i^*. The request that was initially assigned to v_j becomes unassigned.

Two remarks can be emphasised with regard to the previous description. First, when $\mathcal{R}$ contains all unassigned requests, the auction algorithm is better suited for parallel computations (Bertsekas, 1992). However, it should be assumed that nodes and servers are connected by a full mesh in order for nodes to get individual VM prices in a synchronous way as described in the bidding phase and for VMs to get individual request bids as described in the assignment phase. Obviously, this is not the case in the underlying network where connectivity is much less dense. Second, when requests from the same node bid for VMs in the same server, it is likely that y_{r_i} in equation (9) will be equal to x_{r_i} in equation (8) and therefore the bid in equation (10) will only increase by ϵ, which will significantly slow down the convergence especially when ϵ is too small compared to the average price. This problem is known as the *price war*. In the next sections, we will propose several extensions in order to cope with these two problems.

4.2 Extensions

4.2.1 Contention threshold

The first problem we are going to deal with is the price war. In our problem, the similarity can be at the request level as well as at the VM level. Two requests r_i and r_i' are similar if $a_{r_i v_j} = a_{r_i' v_j}$, and two VMs v_j and v_j' are similar if $a_{r_i v_j} = a_{r_i v_j'}$. The set of requests similar to r_i is called the similarity class of r_i and the set of VMs similar to v_j is called the similarity class of v_j. We will assume that no differentiation is applied and therefore, all requests from the same node i form a similarity class denoted $\mathcal{N}(i)$ and all VMs in the same server j form a similarity class as well, denoted $\mathcal{S}(j)$.

The solution proposed in Bertsekas (1992) is to replace the price p_{v_j} of each VM v_j with a contention threshold $\hat{p}_{v_j}$. All VMs in the same server will have the same price

$$p_{v_j} = \min_{u_l \in \mathcal{S}(j)} \hat{p}_{u_l}, \tag{12}$$

but eventually different contention thresholds.

Initially, the contention threshold of each VM v_j is equal to its price p_{v_j}. When v_j is assigned to a request r_i, its contention threshold is increased in such a way that r_i can find an equally attractive VM in a different similarity class, thus avoiding multiple iterations with small ϵ increments prescribed by the original version. Actually, equations (7)–(11) are now calculated according to the contention thresholds, instead of the prices. The other important change is made in equation (9) where the second best net value is sought in another similarity class.

$$y_{r_i} = \max_{\substack{\{l=1,\dots,S\} \\ \{u=1,\dots,c_l\} \\ u_l \notin \mathcal{S}(j*)}} \{a_{r_i u_l} - \hat{p}_{u_l}\}. \tag{13}$$

For the similarity at the request level, we will later propose a method of bundling that will significantly reduce the size of the problem, and consequently its effect on convergence.

4.2.2 Limited network connectivity

There have been a lot of discussions about the practical ways to implement the auction algorithm, especially those related to the synchronisation of processing and data collection. In Bertsekas and Castanon (1991), we find a comparison of synchronous and asynchronous implementations of the algorithm on a shared memory machine. However, as raised in Zavlanos et al. (2008), using a shared memory involves in practice a full mesh underlying network. A new implementation that considers different levels of network connectivity is then proposed. We will briefly describe the extension proposed by Zavlanos et al. (2008) and show how we can use it to solve our problem.

The basic idea of this extension is to let each node make bidding decisions based on a possibly outdated information and make adjustments as it receives updates from neighbouring nodes. In fact, it is assumed that the nodes are interconnected via a network whose connectivity is variable. Thus, the set of VM prices seen by one node could potentially be different from those seen by another, which makes a major difference compared to a shared memory approach.

Let the nodes be connected through an overlay network $O = (N, V)$ where N is the set of nodes and V is the set of undirected overlay links connecting any pair (i, k) of nodes. An overlay link might be a path in the underlaying internet network. Two nodes i and k are neighbours if the overlay link (i, k) exists. Every node i maintains a set of adjacency $\mathcal{N}_i = \{k \in N | (i, k) \in V\}$. Although overlay connectivity depends on the underlay routing, we will assume that the adjacency sets will remain unchanged.

Basically, every node i should associate and manage a set of bidding variables for every request r_i at any time t. These variables include an assignment $v_{r_i}(t)$, a specific view of the contention thresholds $\hat{p}_{r_i v_j}(t)$ and the indexes of the highest bidders $b_{r_i v_j}(t)$ for all VMs v_j in all servers $j \in \{1, ..., S\}$.

We assume that every node i shares bidding information with its directly connected neighbours on periodic intervals of length T using a specific control protocol. When node i receives bidding information at period nT, noted in the following n for simplicity, it updates the bidding information for every request r_i as follows (Zavlanos et al., 2008):

$$\hat{p}_{r_i v_j}(n+1) = \max_{k \in \mathcal{N}_i} \{\hat{p}_{r_i v_j}(n), \hat{p}_{r_k v_j}(n)\} \tag{14}$$

$$b_{r_i v_j}(n+1) = \max_{k \in \arg\max_{z \in \mathcal{N}_i} \{\hat{p}_{r_i v_j}(n), \hat{p}_{r_z v_j}(n)\}} \{b_{r_k v_j}(n)\}. \tag{15}$$

Expression (14) is straightforward. If node i learns that queries on neighbouring nodes have higher contention thresholds for some VM v_j, it will update the corresponding bidding information for its requests. Expression (15) allows every request to simply keep the largest index bidder for every VM v_j in order to break up possible ties.

The assignment of a request r_i is updated if $\hat{p}_{r_i v_{r_i}(n)}(n) \leq \hat{p}_{r_i v_{r_i}(n)}(n+1)$ and $b_{r_i v_{r_i}(n)}(n+1) \neq r_i$. The new assignment is given by

$$v_{r_i}(n+1) = \arg\max_{1 \leq k \leq S} \{a_{r_i v_k} - \hat{p}_{r_i v_k}(n+1)\}, \tag{16}$$

and the new bidder $b_{r_i v_{r_i}(n+1)}(n+1)$ for VM $v_{r_i}(n+1)$ is now r_i. The contention threshold for VM $v_{r_i}(n+1)$ is increased as follows

$$\hat{p}_{r_i v_{r_i}(n+1)}(n+1) = \hat{p}_{r_i v_{r_i}(n+1)}(n) + \gamma_{r_i}, \tag{17}$$

where $\gamma_{r_i} \geq \epsilon$ is the bid increment which satisfies the $\epsilon-$complementary slackness condition (6).

4.2.3 Bundling

The last problem we are going to deal with is scalability. Indeed, the processing of the requests individually could make the size of the problem quite large. This is why, we propose to group requests and VMs into bundles of size σ and to assign one bundle of consolidated VMs to one bundle of requests. This technique has been widely used in the literature. Player grouping is based on social constraints as discussed by Schiller et al. (2019), and is primarily done through a matchmaking process (Guan et al., 2017).

We note $\bar{r}_i$ a bundle of σ requests in node i, and $\bar{v}_j$ a bundle of σ VMs on server j.

Algorithm 1 Distributed virtual machine placement

Given a set of adjacency $\mathcal{N}_i = \{k \in N | (i,k) \in V\}$, every node i maintains for every bundle of requests $\bar{r}_i$ at any period of time nT:

- an assignment $\bar{v}_{\bar{r}_i}(n)$
- a specific view of the contention thresholds $\hat{p}_{\bar{r}_i \bar{v}_j}(n)$
- the indexes of the highest bidders $b_{\bar{r}_i \bar{v}_j}(n)$ for all bundles of VMs $\bar{v}_j$ in all servers $j \in \{1, ..., S\}$.

```
1: procedure BIDDING
2:     $\hat{p}_{\bar{r}_i \bar{v}_j}(n+1) = \max_{k \in \mathcal{N}_i} \{\hat{p}_{\bar{r}_i \bar{v}_j}(n), \hat{p}_{\bar{r}_k \bar{v}_j}(n)\}$    ▷ update contention thresholds
3:     $b_{\bar{r}_i \bar{v}_j}(n+1) = \max_{k \in \arg\max_{z \in \mathcal{N}_i} \{\hat{p}_{\bar{r}_i \bar{v}_j}(n), \hat{p}_{\bar{r}_z \bar{v}_j}(n)\}} \{b_{\bar{r}_k \bar{v}_j}(n)\}$    ▷ update highest bidders
4: end procedure
5: procedure ASSIGNMENT
6:     if $(\hat{p}_{\bar{r}_i \bar{v}_{\bar{r}_i}(n)}(n) \leq \hat{p}_{\bar{r}_i \bar{v}_{\bar{r}_i}(n)}(n+1)$ && $b_{\bar{r}_i \bar{v}_{\bar{r}_i}(n)}(n+1) \neq \bar{r}_i)$ then
7:         $\bar{v}_{\bar{r}_i}(n+1) \in \arg\max_{1 \leq k \leq S} \{a_{\bar{r}_i \bar{v}_k} - \hat{p}_{\bar{r}_i \bar{v}_k}(n+1)\}$    ▷ new assignment
8:         $b_{\bar{r}_i \bar{v}_{\bar{r}_i}(n+1)}(n+1) := \bar{r}_i$    ▷ new bidder for VM $\bar{v}_{\bar{r}_i}(n+1)$
9:         $\hat{p}_{\bar{r}_i \bar{v}_{\bar{r}_i}(n+1)}(n+1) := \hat{p}_{\bar{r}_i \bar{v}_{\bar{r}_i}(n+1)}(n) + \gamma_{\bar{r}_i}$    ▷ update the contention threshold
10:    else
11:        $\bar{v}_{\bar{r}_i}(n+1) = \bar{v}_{\bar{r}_i}(n)$
12:    end if
13: end procedure
```

4.3 DVMP algorithm description

In this section, we will summarise all the above extensions to describe our distributed auction algorithm for aggregated request assignment (DVMP).

Algorithm 1 describes the main iteration of DVMP. The algorithm stops when the contention thresholds do not change after a given number of iterations typically equals to the maximum path length between every pair of nodes.

5 Performance evaluation

In this section, we will study the performance of DVMP in terms of convergence time and optimality gap as a function of the problem size and the network topology. The prices and the contention thresholds are chosen randomly; the results presented in the following studies are averaged over 100 simulations.

In order to set the size of the bundles, we will take a very simple approach based either on the typical capacity of the rendering servers or on the size of a group of players. Considering the typical characteristics of a server with 16 cores and 128 GB of memory and considering that a gaming VM should have at least 1 core and on average 8 GB of memory, we can deduce that a server could support 16 gaming VMs and therefore a bundle could have the size of 16. Schiller et al. (2019) showed that the players' group size varies between 0 and 100 members. Thus, we suggest that a bundle of 15 will be a very reasonable choice.

The problem size depends on the number of nodes and the average number of requests handled by each node. In Table 1, we provide the total number of duplicated nodes, which correspond to individual requests, associated with every network size expressed in number of nodes. As we can see, the size of the assignment problem without bundling quickly becomes quite large if more than a hundred nodes each processes only 15 queries on average. However, when requests are aggregated into bundles of 15, the number of duplicated nodes becomes reasonable again. As in Zavlanos et al. (2008), we choose three types of topology: line, random and complete.

Table 1 Total number of duplicated nodes for each network size

Number of nodes	*Total number of duplicated nodes*	
	Without bundling	*With bundling $\sigma = 15$*
10	349	22
20	614	40
30	915	59
40	1,201	79
50	1,527	100
60	1,820	119
70	2,144	139
80	2,467	159
90	2,739	179
100	3,058	200

5.1 Convergence time

In Figure 4, it is shown that the convergence time increases exponentially for the three topologies with the network size. Obviously, the line topology provides the slowest convergence since it takes N iterations for an update of a node at one end of the network to reach the other end. The complete topology provides the best performance since it virtually simulates a shared memory.

Figure 4 Convergence time of DVMP without bundling (see online version for colours)

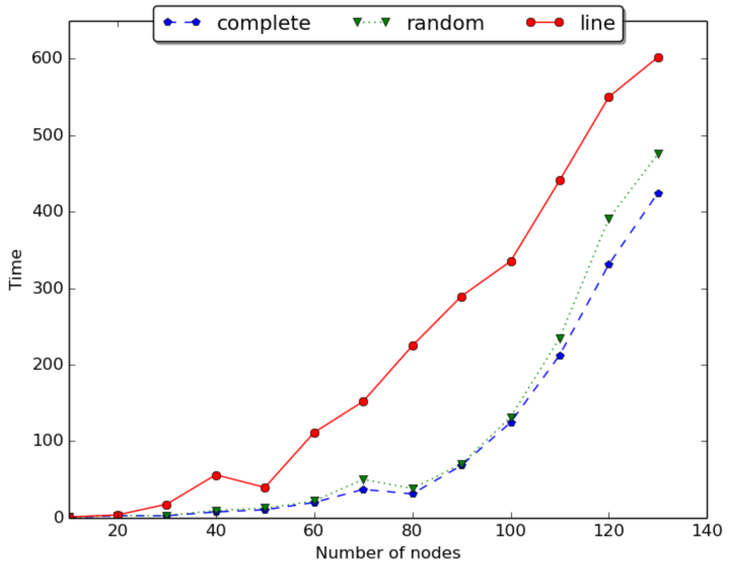
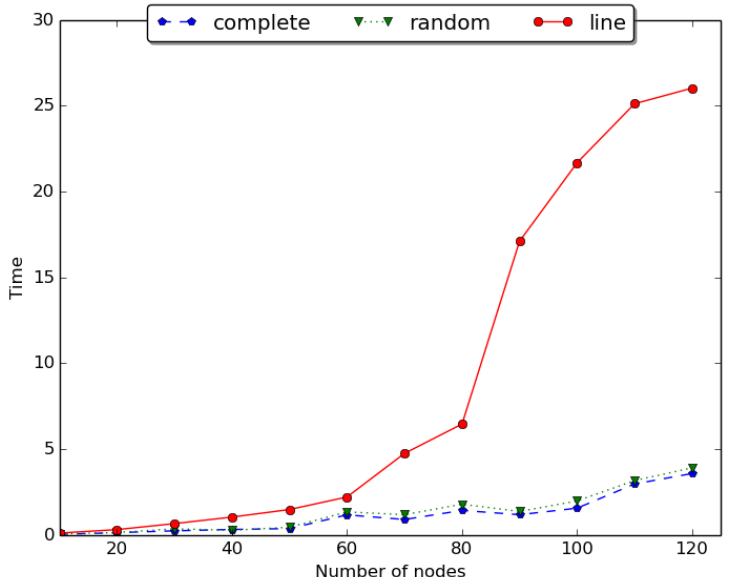


Figure 5 Convergence time of DVMP with bundling (σ = 15) (see online version for colours)



When bundling is applied with bundles of 15 requests, the convergence time is dramatically reduced as shown in Figure 5. The random topology provides a very interesting reduction for large-sized networks. For example, with a network of 120 nodes, the convergence time is reduced from 380 sec to only 3.8 sec. In Table 2, we present the convergence time for the complete topology. The assignment problem is solved with DVMP after a few seconds on a simple PC. Actually, if bundling is applied on each node, only few bundles of requests could be formed. Therefore, the number of duplicated nodes is dramatically reduced and our DVMP algorithm will be closer to a native auction algorithm. Furthermore, it is also possible to define request classes and perform request differentiation so that a fixed number of bundles with variable sizes are formed. In the next section, we will discuss the impact of bundling on the quality of the solution obtained with DVMP.

Table 2 Comparison between the convergence time without and with bundling for complete topology

Number of nodes	*Convergence time*	
	Without bundling	*With bundling σ = 15*
10	0.21	0.035
20	2.69	0.133
30	2.43	0.24
40	7.47	0.32
50	10.23	0.37
60	19.8	1.17
70	36.89	0.9
80	31.07	1.43
90	68.26	1.18
100	124.28	1.56
110	212.53	2.97
120	331.05	3.58
130	424.54	4.84

5.2 *Optimality gap*

We define the optimality gap as the relative difference between the solution z^D found with DVMP, and the optimal solution z^* calculated with a linear programming solver. We used the SCIP optimisation suite (Gamrath et al., 2016) to get the optimal solution. The optimality gap is computed as follows: $Gap = \frac{z^* - z^D}{z^*}$.

In Figure 6, we plot the optimality gap as a function of the number of nodes. Overall, DVMP provides the better solution quality on a complete topology and the gap increases with the network size. The optimality gap is as low as 2% for a complete network of ten nodes. Then, the gap reaches 16% for a network of 100 nodes serving 3,058 requests. Note that on a line network, the performance is also quite good where the gap varies between 6 and 17%. Even though we are using the finest granularity, our algorithm may miss the optimal solution because of the ϵ-complementary slackness condition (6). In our simulations, we used $\epsilon = 0.01$.

When request bundling is used, the performance of DVMP is improved for all topologies as shown if Figure 7. Actually, the optimality gap for a complete topology is reduced to only 9% for a network of 100 nodes. For a random topology, the performance is also very interesting and more stable with an optimality gap varying between 6 and 10% when the network size varies between 10 and 100 nodes. Obviously, the number of duplicated nodes corresponding to request bundles is dramatically reduced with bundling.

Figure 6 Optimality gap of DVMP without bundling (see online version for colours)

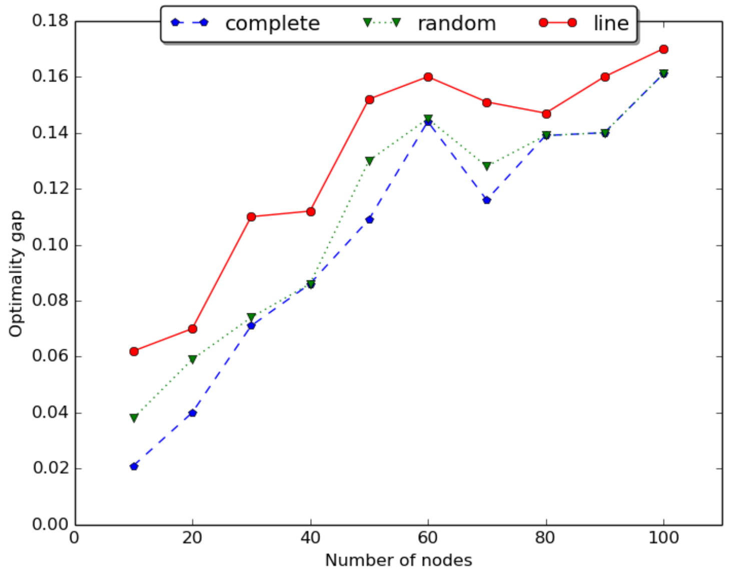
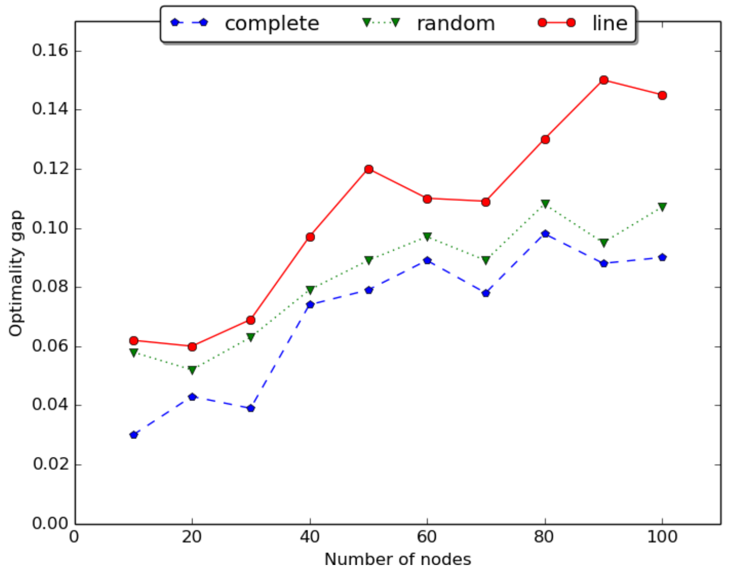


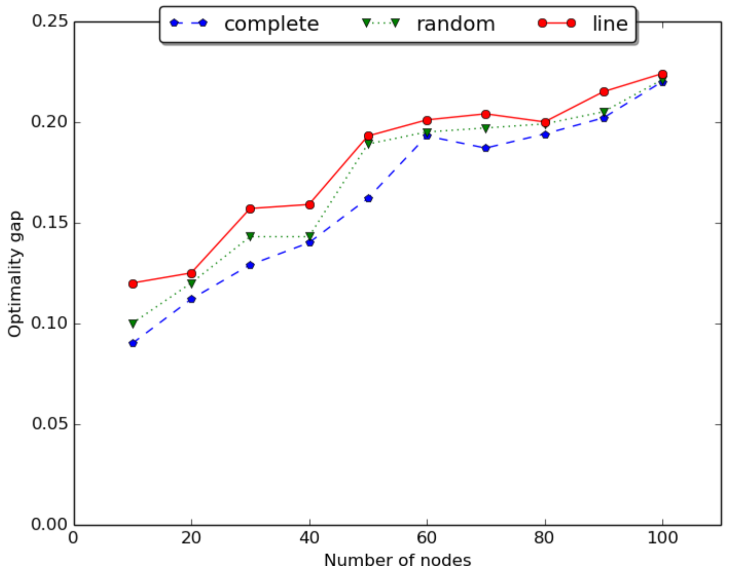
Figure 7 Optimality gap of DVMP with bundling (σ = 15) (see online version for colours)



We can conclude that the overall performance in terms of convergence time and optimality gap for a network with 100 nodes and a random topology is completely satisfactory. Actually, DVMP could solve this problem with bundling in only 4 s and incurring a very low optimality gap of 10%.

Our final study aims to investigate the effect of the bundling rate increase on the overall performance of DVMP. In figure 8, we provide the optimality gap for a bundling rate $\sigma = 30$. We note that the performance of DVMP is degraded using this rather coarse granularity. Consequently, we can say that a judicious choice of the bundling rate could give an acceptable trade-off between the convergence time and the optimality gap. In the scenarios used in our simulations, $\sigma = 15$ provided the best performance.

Figure 8 Optimality gap of DVMP with bundling ($\sigma = 30$) (see online version for colours)



6 Conclusions

In this paper, we considered the problem of assigning VMs hosted on rendering servers in distributed DCs to requests from online gaming players. We introduce an intermediate overlay network of portal nodes between the player and the rendering servers to solve the problem. First, we modelled this problem as a transportation problem where nodes are origins and the servers are destinations. Then, we duplicated the nodes in as many copies as there are requests and formulated the problem as an assignment between the duplicated nodes and the VMs. We used a distributed version of the auction algorithm to solve this finest granularity formulation of our assignment problem. Then, we used a very simple request differentiation to construct bundles of requests and reduce the problem size. Finally, we gathered all the pieces of the distributed and bundled implementations of the bidding and assignment phases to propose our DVMP algorithm. The performance of DVMP is measured using two metrics, namely the convergence time and the optimality gap while varying the topology of the overlay network of nodes. We found that DVMP could solve the assignment problem with bundling in only 4 s and incurring a very low optimality gap of 10%. In a future work, we are investigating request differentiation with more complex criteria such as game type, bandwidth requirements, delay bounds and load balancing.

References

Alicherry, M. and Lakshman, T. (2012) 'Network aware resource allocation in distributed clouds', in *2012 Proceedings IEEE INFOCOM*, pp.963–971.

Bertsekas, D.P. (1992) 'Auction algorithms for network flow problems: a tutorial introduction', *Computational Optimization and Applications*, Vol. 1, No. 1, pp.7–66.

Bertsekas, D.P. and Castanon, D.A. (1991) 'Parallel synchronous and asynchronous implementations of the auction algorithm', *Parallel Computing*, September, Vol. 17, Nos. 6–7, pp.707–732.

Boujelben, Y., Girard, A. and Grégoire, J-C. (2006) 'A sequential algorithm for constructing delay-constrained multirings for multipoint-to-multipoint communications', *Telecommunication Systems*, Vol. 31, No. 1, pp.43–59 [online] https://doi.org/10.1007/s11235-006-5522-1.

Cai, W., Chen, M. and Leung, V.C. (2014) 'Toward gaming as a service', *IEEE Internet Computing*, Vol. 18, No. 3, pp.12–18.

Chaisiri, S., Lee, B-S. and Niyato, D. (2009) 'Optimal virtual machine placement across multiple cloud providers', in *2009 IEEE Asia-Pacific Services Computing Conference (APSCC)*, pp.103–110.

Choy, S., Wong, B., Simon, G. and Rosenberg, C. (2012) 'The brewing storm in cloud gaming: a measurement study on cloud to end-user latency', *11th Annual Workshop on Network and Systems Support for Games (NetGames)*, pp.1–6.

Choy, S., Wong, B., Simon, G. and Rosenberg, C. (2014) 'A hybrid edge-cloud architecture for reducing on-demand gaming latency', *Multimedia Systems*, Vol. 20, No. 5, pp.503–519.

Dammak, M., Andriyanova, I., Boujelben, Y. and Sellami, N. (2018) 'Routing and network coding over a cyclic network for online video gaming', *IEEE Communications Letters*, Vol. 22, No. 6, pp.1188–1191.

D'Angelo, G., Ferretti, S. and Marzolla, M. (2015) *Cloud for Gaming*, arXiv preprint arXiv:1505.02435.

Deng, Y., Li, Y., Seet, R., Tang, X. and Cai, W. (2018) 'The server allocation problem for session-based multiplayer cloud gaming', *IEEE Transactions on Multimedia*, Vol. 20, No. 5, pp.1233–1245.

Gamrath, G., Fischer, T., Gally, T., Gleixner, A.M., Hendel, G., Koch, T., Maher, S.J., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schenker, S., Schwarz, R., Serrano, F., Shinano, Y., Vigerske, S., Weninger, D., Winkler, M., Witt, J.T. and Witzig, J. (2016) *The SCIP Optimization Suite 3.2*, Technical Report 15-60, ZIB, Takustr. 7, 14195 Berlin.

Guan, Y., Deng, Y. and Tang, X. (2017) 'On matchmaking for multiplayer cloud gaming', in *2017 15th Annual Workshop on Network and Systems Support for Games (NetGames)*, pp.1–3.

Han, Y., Guo, D., Cai, W., Wang, X. and Leung, V. (2020) 'Virtual machine placement optimization in mobile cloud gaming through qoe-oriented resource competition', *IEEE Transactions on Cloud Computing*, p.1.

Hao, F., Kodialam, M., Lakshman, T.V. and Mukherjee, S. (2017) 'Online allocation of virtual machines in a distributed cloud', *IEEE/ACM Transactions on Networking*, Vol. 25, No. 1, pp.238–249.

Hong, H.J., Chen, D.Y., Huang, C.Y., Chen, K.T. and Hsu, C.H. (2015) 'Placing virtual machines to optimize cloud gaming experience', *IEEE Transactions on Cloud Computing*, Vol. 3, No. 1, pp.42–53.

Jarschel, M., Hoßfeld, T., Scheuring, S. and Schlosser, D. (2011) 'An evaluation of qoe in cloud gaming based on subjective tests', *Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pp.330–335.

Kessaci, Y., Melab, N. and Talbi, E-G. (2013) 'A Pareto-based genetic algorithm for optimized assignment of VM requests on a cloud brokering environment', in *2013 IEEE Congress on Evolutionary Computation*, pp.2496–2503.

Laghrissi, A. and Taleb, T. (2019) 'A survey on the placement of virtual resources and virtual network functions', *IEEE Communications Surveys Tutorials*, Vol. 21, No. 2, pp.1409–1434.

Li, J., Zhu, Y., Yu, J., Long, C., Xue, G. and Qian, S. (2018) 'Online auction for IaaS clouds: towards elastic user demands and weighted heterogeneous VMs', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 29, No. 9, pp.2075–2089.

Lin, Y. and Shen, H. (2017) 'CloudFog: leveraging fog to extend cloud gaming for thin-client MMOG with high quality of service', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 2, pp.431–445.

Masdari, M., Nabavi, S.S. and Ahmadi, V. (2016) 'An overview of virtual machine placement schemes in cloud computing', *Journal of Network and Computer Applications*, Vol. 66, pp.106–127 [online] https://www.sciencedirect.com/science/article/pii/S1084804516000291.

Meng, X., Pappas, V. and Zhang, L. (2010) 'Improving the scalability of data center networks with traffic-aware virtual machine placement', in *2010 Proceedings IEEE INFOCOM*, pp.1–9.

Parida, S. and Pati, B. (2020) 'A cost efficient service broker policy for data center allocation in IaaS cloud model', *Wirel. Pers. Commun.*, Vol. 115, No. 1, pp.267–289 [online] https://doi.org/10.1007/s11277-020-07570-1.

Peake, J., Amos, M., Costen, N., Masala, G. and Lloyd, H. (2022) 'PACO-VMP: parallel ant colony optimization for virtual machine placement', *Future Generation Computer Systems*, Vol. 129, pp.174–186 [online] https://www.sciencedirect.com/science/article/pii/S0167739X21004568.

Samimi, P., Teimouri, Y. and Mukhtar, M. (2016) 'A combinatorial double auction resource allocation model in cloud computing', *Information Sciences*, Vol. 357, pp.201–216 [online] https://www.sciencedirect.com/science/article/pii/S0020025514001054.

Saxena, D., Gupta, I., Kumar, J., Singh, A.K. and Wen, X. (2021) 'A secure and multiobjective virtual machine placement framework for cloud data center', *IEEE Systems Journal*, pp.1–12.

Schiller, M.H., Wallner, G., Schinnerl, C., Calvo, A.M., Pirker, J., Sifa, R. and Drachen, A. (2019) 'Inside the group: investigating social structures in player groups and their influence on activity', *IEEE Transactions on Games*, Vol. 11, No. 4, pp.416–425.

Shea, R., Liu, J., Ngai, E.C.H. and Cui, Y. (2013) 'Cloud gaming: architecture and performance', *IEEE Network*, Vol. 27, No. 4, pp.16–21.

Soliman, O., Rezgui, A., Soliman, H. and Manea, N. (2013) 'Mobile cloud gaming: issues and challenges', *International Conference on Mobile Web and Information Systems*, pp.121–128.

Tan, X., Leon-Garcia, A., Wu, Y. and Tsang, D.H.K. (2020) 'Online combinatorial auctions for resource allocation with supply costs and capacity limits', *IEEE Journal on Selected Areas in Communications*, Vol. 38, No. 4, pp.655–668.

Zaman, S. and Grosu, D. (2013) 'Combinatorial auction-based allocation of virtual machine instances in clouds', *Journal of Parallel and Distributed Computing*, Vol. 73, No. 4, pp.495–508 [online] https://www.sciencedirect.com/science/article/pii/S0743731512002870.

Zavlanos, M.M., Spesivtsev, L. and Pappas, G.J. (2008) 'A distributed auction algorithm for the assignment problem', *47th IEEE Conference on Decision and Control*, pp.1212–1217.

Zhang, J., Huang, H. and Wang, X. (2016) 'Resource provision algorithms in cloud computing: a survey', *Journal of Network and Computer Applications*, Vol. 64, pp.23–42 [online] https://www.sciencedirect.com/science/article/pii/S1084804516000643.

Zhang, J., Yang, X., Xie, N., Zhang, X., Vasilakos, A.V. and Li, W. (2020) 'An online auction mechanism for time-varying multidimensional resource allocation in clouds', *Future Generation Computer Systems*, Vol. 111, pp.27–38 [online] https://www.sciencedirect.com/science/article/pii/S0167739X19331127.

Zhang, Y., Jiao, L., Yan, J. and Lin, X. (2019) 'Dynamic service placement for virtual reality group gaming on mobile edge cloudlets', *IEEE Journal on Selected Areas in Communications*, Vol. 37, No. 8, pp.1881–1897.