UNIVERSITY OF SOUTHAMPTON

Faculty of Physical Engineering and Science
School of Electronics and Computer Science

A project report submitted for the award of
MEng Computer Science

Supervisor: Dr Tim Norman

Reinforcement learning agents for online elastic resource allocation in cloud computing

by **Mark Towers**

April 25, 2020

University of Southampton

Abstract

Faculty of Physical Engineering and Science
School of Electronics and Computer Science

A project report submitted for the award of MEng Computer Science

Reinforcement learning agents for online elastic resource allocation in cloud computing

by Mark Towers

Edge clouds enable computational tasks to be completed at the edge of the network, without relying on access to remote data centres. A key challenge in these settings is that server have limited computational resources often need to be allocated to many self-interested users. Here, existing resource allocation approaches usually assume that tasks have inelastic resource requirements (i.e., a fixed amount of compute time, bandwidth and storage), which may result in inefficient resource use. In this paper, we expand previous work, that utilises an elastic resource requirement mechanism, to an online setting such that job will arrive over time with the task prices and resource allocation determined through agents trained using reinforcement learning.

Contents

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a degree at this University;
2. Where any part of this thesis has previously been submitted for any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as: S.R. Gunn. Pdflatex instructions, 2001. URL `http://www.ecs.soton.ac.uk/~srg/softwaretools/document/`
   C. J. Lovell. Updated templates, 2011
   S.R. Gunn and C. J. Lovell. Updated templates reference 2, 2011

Signed:... Date:..................

Acknowledgements

I want to thank my parents for all of the support they have given me because I would not be where I am now without them. Also to my housemates for surviving with me pestering them about proof reading badly written papers and for dealing with me stressing about this project.

This project wouldn't have started without Dr Sebastian Stein, Professor Tim Norman and a team of Pennsylvania State University that has produced a paper investigating the static case of this problem. Thank to all of them for sharing ideas and support for that paper and this project.

Also to Professor Tim Norman for this constant guidance over the project and the wisdom to know what for me to investigate and implement.

Chapter 1

Project problem

Cloud computing is a rapidly growing service with competition from Google, Amazon, Microsoft and more that aims to allow users to run computer programs that are too large, difficult or time consuming for users to run locally. These services provide the computational resources, e.g. cpu cores, RAM, hard drive space, bandwidth, etc to be able to run such programs. However, as these resources are limited, bottlenecks can occur when numerous users all require large amounts of these resources, limiting the number of tasks [1] that can be run on the cloud servers simultaneously.

For Google Cloud Services (GCP), Microsoft Azure or Amazon Web Services, their cloud computing facilities contain huge server nodes limiting the probability of such a bottleneck from occurring. And if such an event does occur users also have a range of data centres across the global to use instead if a single data centre does becomes overloaded. Therefore this work considers still a developing paradigm (Mao et al., 2017) called Edge/Mobile cloud computing. Edge cloud computing is believed to have a wide range of application where traditional cloud computing would be impractical. This could be due to tasks being highly delay-sensitive, intermittent internet connectivity or high operational security that prevent or limit the effectiveness of traditional cloud computing.

Currently disaster response (Guerdan et al., 2017), smart cities (Alazawi et al., 2014) and internet-of-things (Corcoran and Datta , 2016) are all area that utilise edge cloud computing due to its ability to process computationally small tasks

[1]Tasks, Programs and Jobs will be used interchangeable to refer to the same idea of a computer programs that has a fixed amount of resources required to compute.

locally with low latency. For example, in smart cities, this allows for smart intersection systems using of road-side sensors or smart traffic lights based on cameras to minimise the waiting times (Mustapha et al., 2018). Or for the police to analysis CCTV footage to spot suspicious behaviour or to track people between cameras (Sreenu and Saleem Durai, 2019). In the case of disaster response, maps can be produced using data from autonomous vehicles sensors that can then be used in the search for potential victims and support responders (Alazawi et al., 2014).

However the problem of bottlenecking is of particular relevant in edge cloud computing, as instead of large server farms that can be geographically distant from the users. Edge cloud computing server are significantly smaller, possibly just high powered desktop computers and single server nodes. This results in greater demand on server resources, meaning that efficient allocation of resources is extremely important. Because of this, resource allocation in edge cloud computing is an important and interesting research area within edge cloud computing.

However it is believed that there are shortcoming in existing work about resource allocation within edge cloud computing (Farhadi et al., 2019; Bi et al., 2019) due to the nature of how task resource usage is determined. Traditionally, a user would submit a request for a fixed amount of resources, i.e. 2 cpu cores, 8Gb of ram, 20Gb of storage, that would be allocated for the user. As a result, these resources can be redistributed till the user finishes with them. This resource allocation mechanism is effective for cloud computing due to its simplicity for the user for deciding resource requirements, can utilise a linear pricing mechanisms and services having large resource pool making bottlenecking rare.

Therefore in previous work by this author (Towers et al., 2020) a novel resource allocation mechanism was proposed to allow for significantly more flexibility in determining resource usage with the aims of reducing possible bottlenecking. Mechanism is based on the principle that the time taken for an operation to complete is generally proportional to the resources provided for the operation. An example for this is downloading an image, the time taken is proportional to the bandwidth allocated. This sort of flexibility is similarly true for computing a task [2] or sending results back to the user as well. Then by requesting that users

[2] This is not always true for all computational tasks, however for this work we presume this and leave this case to future work

provide the task's total resource usage over its lifetime instead of the requested resource usage. Using this information task resource usage is determined by the server rather than the user. Therefore using a deadline, provided by the user, it is possible to reallocate resources around tasks to reduce the overall strain on certain resources while still finishing the task with its deadline. Using this alternative resource allocation mechanism, it was found that results could achieve 20% better than traditional resource allocation mechanisms in static cases investigated in Towers et al. (2020). This is due to the ability to proper balancing of resources, preventing bottlenecks occurring as often, that in turn allowed more tasks to run simultaneously and to reduce the price for user to run a task.

But in this previous work (Towers et al., 2020), the flexible resource allocation mechanism was only considered in a static or one-shot approach where all tasks were presented at the first time step. At which point tasks would be auctioned and resource allocated. As a result, practically the proposed algorithms would require tasks to be processed in batches such that servers would bid on all tasks submitted every 5 minutes for example. Therefore previous work could also not dynamically change the resources allocated between batches making it impractical to be used commercially, this work aims to address these problems in previous work.

These problems are addressed by introducing time into the optimisation problem (outlined in section 3.1) but due to this addition, all previous mechanism proposed in Towers et al. (2020) are incompatible with the new optimisation problem. Therefore using a standard auction mechanism, this project investigates different methods of learning how to bid on tasks based on their resource requirements and to efficiently allocate resources to tasks by a server.

This report is set out is following chapters. Chapter 3 proposed a solution to the project outline in this chapter with chapter 4 justifying why this approach as taken over alternative. Chapter 2 investigates the previous research that this project builds upon within both resource allocation in cloud computing and reinforcement learning methods. The proposed solution is then implemented in chapter 5 with testing and evaluation in chapters 6 and 7 respectively.

In addition to this report, the paper referred to as Towers et al. (2020), with a copy in appendix 4, was completed within this academic year and thus consider part of this project's work. In addition to this paper, the work was presented at SPIE Defense + Commercial Sensing 2020 as a recorded digital presentation with a copy of the slides in appendix 4 and a link provided for the recording.

Chapter 2

Related Work

There is a considerable amount of research in the area of pricing and resource allocation in cloud computing, of which some use auction mechanisms to deal with competition (Kumar et al., 2017; Zhang et al., 2017; Bingqian Du, 2019; Bi et al., 2019) as this project does. Therefore section 2.1 presents the similar approaches to resource allocation in cloud computing and edge cloud computing to the one taken in this project.

The proposed solution of the project (presented in chapter 3) uses a form of machine learning, called reinforcement learning. Section 2.2 explores the research and the current state of the art algorithms of deep Q learning and policy gradient that are used in this project.

2.1 Related Work in Cloud Computing

A majority of approaches for pricing and resource allocation in cloud computing use a fixed resource allocation mechanism such that user request a fixed amount of certain resource from the cloud provider. However this mechanism, as previously explained, provides no control over the resources quantity allocated to the server but means that no specially resource allocation mechanism is needed. Therefore a majority of research has focused on designing efficient and incentive compatible auction mechanism. A survey of these approaches for double auction mechanism in cloud computing is outlined by Kumar et al. (2017), (Kumar et al., 2017; Zhang et al., 2017; Bingqian Du, 2019; Bi et al., 2019).

Other closely related work on resource allocation in edge clouds Farhadi et al. (2019) considers both the placement of code/data needed to run a specific task, as well as the scheduling of tasks to different edge clouds. The goal there is to maximize the expected rate of successfully accomplished tasks over time. Our work is different both in the setup and the objective function. Our objective is to maximize the value over all tasks. In terms of the setup, they assume that data/code can be shared and they do not consider the elasticity of resources.

Previous work by this author in Towers et al. (2020) proposed the novel resource allocation (explained in chapter 1) along with an optimisation problem mathematically describing the resource allocation. The goal of the problem is to maximise the social welfare, the sum of all task values that are run and completed successfully within the task deadline. The paper presents three mechanism for the optimisation problem, one to maximise the social welfare and two auction mechanisms. A greedy algorithm presented allows for quick approximation of a solution through the use of several heuristics in order to maximise the social welfare. Results found that this mechanism could achieves over 90% of the optimal solution given certain heuristics compared to fixed resource allocatiom methods with only 70% of the solution solution. The algorithm is a polynomial time algorithm that will find solution within $\frac{1}{n}$ of the optimal social welfare. The heuristics were for ordering the task by density then for each task, selecting a server based on available resource on each servers then to allocate resources that minimises the final resource heuristics. The first of the auction mechanisms is a novel distributed iterative auction developed using a reverse vcg principle (Vickrey, 1961; Clarke, 1971; Groves, 1973) to calculate a task price. That meant that a task doesnt need to reveal its private value also that the auction could be run in a decentralised way. This means that the auction is budget balanced however it is not economically efficient or incentive compatible. The mechanism achieves over 90% of the optimal solution due to its solving of the optimisation problem allowing it to find optimal resource allocations. The third algorithm is an implementation of a single parameter auctions (Nisan et al., 2007) using the greedy algorithm (explained previously) to find the critical value of a task. Using this mechanism with a monotonic value density heuristic results in the auction being incentive compatible also well a inherienting the social welfare properties of the greedy mechanism.

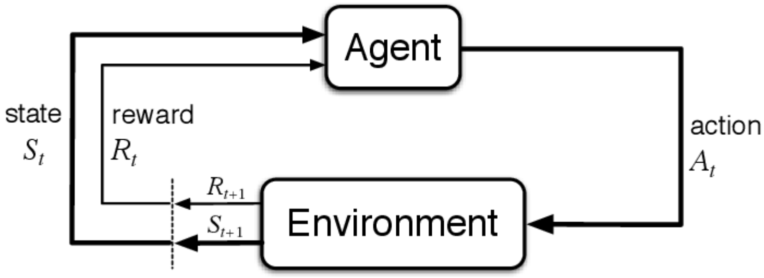


FIGURE 2.1: Reinforcement learning model (Source: Sutton and Barto (2018))

2.2 Related Work in Reinforcement learning

Computer scientists have always been interested in testing computers against humans (Turing, 2009) and a key element of humans is the ability to learn from rewards. For computers this ability is much more complex and researchers have found a variety of ways to allow computers to do this. These methods are broadly grouped into three categories of supervised, unsupervised and reinforcement learning. Supervised learning uses pairs of inputs to true outputs like in case of image classifications where each image has a correct category for the image to be mapped to. Unsupervised learning instead doesn't have a true output meaning that algorithms tries to find links between similar data.

However both of these methods are not effective for real world interactions as agents must make a series of actions that result in rewards. Algorithms designed for these problems with a series of actions fall into the category of reinforcement learning which aims to maximise the agent rewards over time (an example environment has the format 2.1). This is the area of machine learning that this project utilises as the problem can be modelled as a markov decision process (explained in section 3.3) that allows agents to interact with the environment to learn over time. Reinforcement learning is a rapidly growing field of research within AI due to its real world applications like driving a car, playing games, etc.

Q-learning algorithm Watkins and Dayan (1992) is a learning method used for estimating the action-value function, one of the bases on for modern reinforcement learning. As the series of actions can be formed into a tree of actions, an agent is interested in which actions will result in the largest reward in the future. This is formulated as in equation (2.2).

$$Q(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \cdots] \tag{2.1}$$

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2.2}$$

$$\tag{2.3}$$

However the curse of dimensionally was found to be a major problem as to use Q learning required forming a table of state-actions in order to calculate and as the number of dimensions increase, the number of state-actions will increase exponentially. Therefore making the method impractical for problem that had large state space. Therefore use of a function approximate is used to circumvent this problem, traditionally done using a neural network. Work by Mnih et al. (2013) using a deep convolution neural network was able to achieve state of the art in six of seven games tried on atari with three of these scores being superhuman. This work was followed up by Mnih et al. (2015) and found that with no modifications to the hyperparameters, neural network and training method; state of the art results were achieved in almost all 49 atari games and superhuman results in 29 of these games. Additional heuristics have been proposed for deep Q learning: double DQN (van Hasselt et al., 2015), prioritized experience replay (Schaul et al., 2015), dueling network architecture (Wang et al., 2015), multi-step bootstrap targets (Sutton, 1988; Sutton and Barto, 2018), A3C Mnih et al. (2016), distributional Q-learning (Bellemare et al., 2017) and noisy DQN (Fortunato et al., 2017). These methods were combined to together Hessel et al. (2017), called rainbow DQN, achieving over 200% of the original DQN algorithm and over 50% than any optimisation on its own in a quarter of the observations.

Using the base of Q-learning, policy gradient 2.2 separate the action selection policy to the q-value policy. In Q-learning, the selection the action is based on the maximum Q-value of the actions however policy gradient separates. This has the advantages of being able to deal with both discrete and continuous action spaces where Q-learning can only deal with discrete action space. Also the learning method doesn't require ϵ-greedy action selection that can for Q-learning cause the resulting policy to differ from the optimal policy. Therefore policy gradient has been used to master the game of Go (Silver et al., 2017) and achieve top 1% in Dota 2 (OpenAI, 2018) and Starcraft 2 (Vinyals et al., 2017).

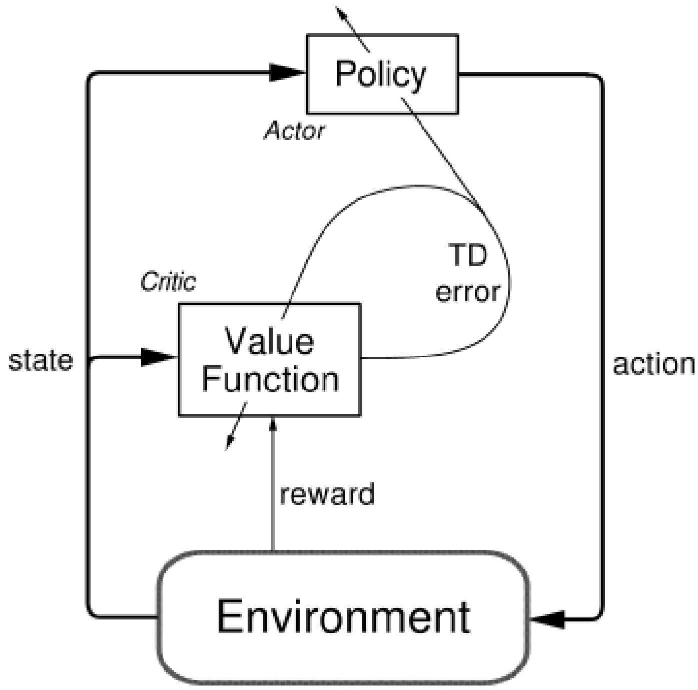


FIGURE 2.2: Actor Critic model (Source: Sutton and Barto (2018))

Chapter 3

Proposed solution to problem

In chapter 1, the problem that this project aims to address was outlined along with a short description of the proposed solution. This chapter builds upon that giving a format mathematical model for the problem (section 3.1). While the overall problem is about resource allocation, we believe that cloud provider would wish to be paid for the use of their services so this additional problem is addressed in section 3.2 for the auctioning of tasks. With the two problems explained in the previous two sections, section 3.3 proposed auction agent (subsection 3.3.1) who can learn how to maximise their profits over time and resource allocation agents (subsection 3.3.2) to determine how to complete all of the tasks.

3.1 Resource Allocation Optimisation problem

Using the flexible resource allocation principle presented in Towers et al. (2020), a format mathematical description of the model can developed. This principle is that for certain resources, the time taken for a operation to occur, e.g. loading of a program, computing the program and sending of results, etc, is proportional to the amount of resources allocated to completed the operation. [1] Therefore instead of allowing the user to requesting a fixed amount of resources for loading, computing and sending back the results of a program, the user would

[1] This principle is not always true, for example, video decompression is a generally a single thread operation and cannot be effectively multi-threaded. However for this project we only consider tasks that can be parallelised effectively.

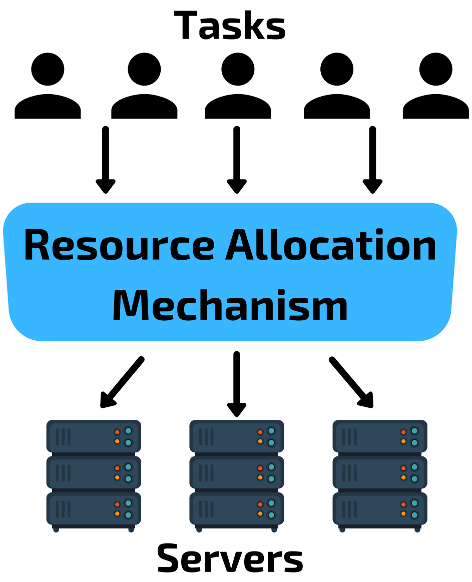


FIGURE 3.1: System model

instead inform the server the total amount of bandwidth required, computational power, etc for the task to be completed. Then the server can dynamically allocate its available resources to all of its allocated tasks. This is believed to be effective compared to standard resource allocation mechanism as bottleneck can occur on certain resource preventing additional tasks from being allocated due servers not having the requested resources available for the task. Therefore using this principle, a modified version of a standard resource allocation formulation can be described to maximise social welfare.

A sketch of the system is shown in Fig. 3.1. We assume that in the system there is a set of $I = \{1, 2, \ldots, |I|\}$ servers are heterogeneous in all characteristics. Each server has a fixed availability of resources: storage for the code/data needed to run a task (e.g., measured in GB), computation capacity in terms of CPU cycles per time interval (e.g., measured in FLOP/s), and communication bandwidth to receive the data and to send back the results of the task after execution (e.g., measured in Mbit/s). We denote these resources for server i: the storage capacity as S_i, computation capacity as W_i, and the communication capacity as R_i.

There is a set $J = \{1, 2, \ldots, |J|\}$ of different tasks that require service from one of the servers in set $I = \{1, 2, \ldots, |I|\}$. To run any of these tasks on a server requires storing the appropriate code/data on the same server. These could be, for example, a set of images, videos or CNN layers in identification tasks. The storage size of task j is denoted as s_j with the rate at which the program is transferred to the server i at time t being $s^{'}_{i,j,t}$. For a task to be computed successfully, it must fetch and execute instructions on a CPU. We consider the total number of CPU cycles required for the program to be w_j, where the rate at which the CPU cycles are assigned to the task on server i at time t is $w^{'}_{i,j,t}$. Finally, after the task is run and the results obtained, the latter need to be sent back to the user. The size of the results for task j is denoted with r_j, and the rate at which they are sent back to the user is $r^{'}_{i,j,t}$ on server i at time t. Every task has a beginning time, denoted by b_j and a deadline, denoted by d_j. This is the maximum time for the task to be completed in order for the user to derive its value. This time includes: the time required to send the data/code to the server, run it on the server, and get back the results. Therefore for the task to be successfully completed, it must completed fulfill the constraint in equation (3.1). These operations must occur in order (loading, computing then sending of results) as a server couldn't start computing a task that was not fully loaded on the machine.

$$\frac{s_j}{\sum_{t=b_j}^{d_j} s^{'}_{i,j,t}} + \frac{w_j}{\sum_{t=b_j}^{d_j} w^{'}_{i,j,t}} + \frac{r_j}{\sum_{t=b_j}^{d_j} r^{'}_{i,j,t}} \leq d_j \qquad \forall j \in J \tag{3.1}$$

As server have limited capacity, the total resource usages for all tasks running on a server must be capped. The storage constraint (equation (3.2)) is unique as the previous amount loaded in kept till the end of a program on server. While the computation capacity (equation (3.3) is the sum of compute used by all of the tasks on a server i at time t and the bandwidth capacity (equation (3.4)) is the sum of loading and sending usages by tasks.

$$\sum_{j \in J} \left(\sum_{t=b_j}^{d_j} s^{'}_{i,j,t} \right) \leq S_i, \qquad \forall i \in I \tag{3.2}$$

$$\sum_{j \in J} w^{'}_{i,j,t} \leq W_i, \qquad \forall i \in I, t \in T \tag{3.3}$$

$$\sum_{j \in J} s^{'}_{i,j,t} + r^{'}_{i,j,t} \leq R_i, \qquad \forall i \in I, t \in T \tag{3.4}$$

$$\tag{3.5}$$

3.2 Auctioning of Tasks

While the mathematically description of the problem presented above doesn't contain any auctioning properties, in real life cloud providers normally wish to be paid to the use of their services. However due to the modifications that this project has to make to the optimisation problems compared to a traditional cloud computing optimisation problem. All traditional auction mechanisms that have been discussed in section 2.1 cannot be used as the user is not requesting a fixed amount of resources nor can the available resources be easily computed as this is dynamic depending on the currently allocated tasks to a server. This means that a novel or modified auction mechanism must be used to deal with these changes. Due to the complexities of devising new auction mechanism and the large corpus of research on auctions already, this project has chosen to use the Vickrey auction (Vickrey, 1961). This decision is justified in section 4.1 on why auction over other alternatives was chosen.

The modification that is made to the Vickrey auction is to reverse the auction as server are bidding on tasks instead of a task paying servers. Because of this, the auction is referenced to as the reverse Vickrey auction. The auction works by allowing servers all submit their bid for the task with the winner being the server with the lowest price but actually only gains second lowest price. The advantage of using the Vickrey auction is that it is incentive compatible meaning that the dominant strategy for bidding on a task is to bid your truthful value. This should help server as they dont need to learn how to outbid another agent as it only needs to consider its own evaluation. This also allows agents to learn through self-play effectively. The auction also has only a single round of bidding compared to alternative auctions like English or Dutch auctions. This makes auctioning fast no matter the number of servers and it also allows for tasks to set a reserve price.

3.3 Proposed Agents

Using the optimisation formulation and auction problem from the previous two sections, the problem can explained using the markov decision process format in figure 3.2. The format separates out the auction parts of the problem and the resource allocation part of the problem with separate agents to act during these parts of the problem. While the state and action spaces of the agents

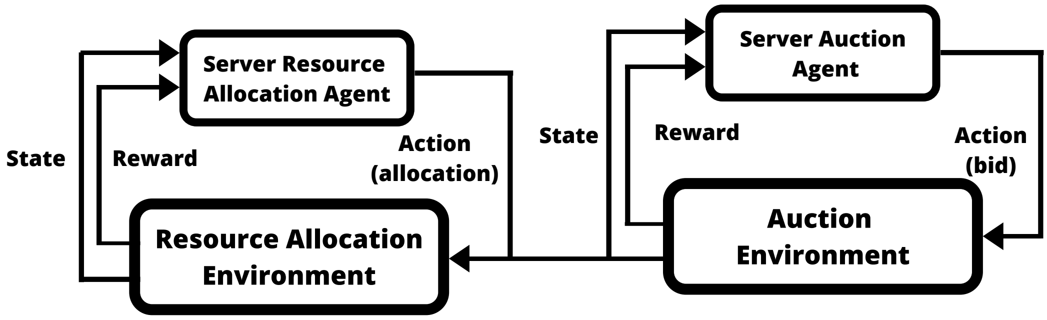


FIGURE 3.2: Markov Decision process system model

can be the same, the policies that the agents will be following are completely different therefore making it impossible to merge the two agents into one. In subsection 3.3.1 and 3.3.2 agents are proposed for the auction environment and resource allocation environment respectively.

The overall problem is of particularly interesting for multi-agent reinforcement learning environment as aim of the environment is cooperative (to maximise the social welfare) but the servers in competition during the auctions in order to maximise their profit. But then the server must be cooperative in allocating of resources to each task as the server wish to complete as many tasks as they can.

3.3.1 Proposed auction agents

Traditionally pricing mechanisms (Al-Roomi et al., 2013) rely on mixture of metrics; resource availability, resource demand, quality of service, task resource requirements, task resource allocation quantity, etc. However these values are difficult to approximate at the point of server bidding. Due to the complexity of a function to this, reinforcement learning will be used to learn this function in order to maximise the profits of the server. Simple heuristics will also be implemented in order compare the effectiveness of the reinforcement learning to untrained heuristics.

As the space of prices, the action space of the agent, is continuous value greater than zero. Therefore a deep deterministic policy gradient (Silver et al., 2014) agent that allows for discrete state spaces and continuous action spaces will be implemented. Also as the action space can be discretized then several deep Q learning agents will be trained that use different heuristics in order to compare the results.

These agents will use neural networks, as they function as universal function approximators (Csáji, 2001), for the agents to use to learn. Because of this, a long/short term memory (LSTM) layer will be used as it allows for multiple inputs and provides a single output that will have several additional layers to allow additional complexity. With the network ending at a single ReLU neuron for DDPG or multiple linear activation neurons for DQN agents. The justification for this network over other neural network models is explained in section 4.2.1

3.3.2 Proposed resource allocation agents

When a new task is allocated to the server or a task completes a stage then the server resource will be redistributed. As the problem of how to allocation resources isn't as complex as the agent pricing in section 3.3.1, both simple heuristics and reinforcement learning agents will be implemented in order to compare effectiveness.

However a similar problem exists to the proposed auction agents (in subsection 3.3.1) as to know how to allocate resources to be single task is be aware of the other tasks that the server currently contains. Therefore a similar network is proposed to allow for the other tasks to be passed in in order to compare the current task having resources allocated to and the other tasks also allocated to the server. The only other change is that the action space doesn't represent the equivalent amount of a certain type of resources but instead the weighting for that resources. This is as if the network wanted to allocate as many resources as possible while having to keep the total amount of resource under the total available then this is extremely difficult. Therefore the action space instead represents the weight of that resource with the equivalent amount of resources can then be determined by the server. This aims to reduce the needed complexity of the resource allocation function and speed up the learning time of the agent.

Chapter 4

Justification of the solution

The proposed solutions for the project is outlined in chapter 3. This chapter explains the reasoning for the chooses made for the solution with regards to the auction mechanism in section 4.1 and the proposed agents in section 4.2.

4.1 Justification of auction mechanisms

In mechanism design, there are numerous types of auctions, each with that have different properties and applications. The types of auction that this project is interested in is single indivisible items where an item is sold as a single units compared to combinatorial auctions. This is as while the item has multiple resource requires, a server is required to buy the task as a single unit with a single price. Table 4.1 outlines a description of possible applicable auction mechanisms while table 4.2 outline the properties of each auction mechanisms in the previous table.

Auction type	**Description**
English auction	A traditional auction where all participant can bid on a single item with the price slowing ascending till only a single participant is left who pays the final bid price. Due to the number of rounds, this requires a large amount of communication and for tasks to be auctioned in series.

Dutch auction	The reverse of the English auction where the starting price is higher than anyone is willing to pay with the price slowly dropping till the first participant ”jumps in”. This can result in sub-optimal pricing if the starting price is not highest enough or to large number of rounds till a bid. Plus due the auctions occurring over the internet, latency can have a large effect on the winner.
Japanese auction	Similar to the English auction except that the auction occurs over a set period of time with the last highest bid being the winner. This means that it has the same disadvantages as the English auction except that there is no guarantee that the price will converge to the maximum. Plus additional factors like latency can have a large effect on the winner and resulting price. But due to time limit, the auction has known amount of time to finish unlike the English or Dutch auctions.
Blind auction	Also known as a First-price sealed-bid auction, all participants submit a single secret bid for an item with the highest bid winning who pays their bid value. As a result there is no dominant strategy (not incentive compatible) as an agent would not wish to bid higher than their task evaluation. But if all other agents bid significantly lower then it would have been beneficial for the agent to bid much lower than their true evaluation. But due to there being only a single round of biding, latency doesn’t affect an agent and many more auctions could occur within the same time compared to the English, Dutch or Japanese auction would take longer to run.
Vickrey auction (Vickrey, 1961)	Also known as a second-price sealed bid auction, instead all participants submit a single secret bid for an item with the highest bid winning. However the winner only pays the price of the second highest bid. Because of this, it is a dominant strategy for an agent to bid its true value as even if the bid is much higher than all other participants its doesn’t matter as they pay the minimum they would want to win.

TABLE 4.1: Descriptions of feasible auctions for the project; English, Dutch, Japanese, Blind and Vickrey auction

Due to the properties that the Vickrey auction has in compared to the other auctions, of incentive compatibility and single round being of most importance

Auction	Incentive compatible	Iterative	Fixed time length
English	False	True	False
Japanese	False	True	True
Dutch	False	True	False
Blind	False	False	True
Vickrey	True	False	True

TABLE 4.2: Properties of the auctions described in Table 4.1

means that this auction is believe the most appropriate for this project. The believed advantage of the auction being incentive compatible is that agent only need to learn the true value of a task compared to the minimum price that the agent believes they could buy the task for. This has the advantage of allowing agents to self-train as the agents don't need to learn how to out price other agents. Another advantage of the auction is that it is not iterative, making the auction fast with only a single round of bidding required. This means that the auction can be certain to complete within a fixed amount of time as server must submit a bid within the time frame or loss out on the task.

4.2 Justification of Auctioning and Resource allocation agent

There are a range of possibly neural network layers that allow for multiple inputs, table 4.3 outlines the most common of these layers. This information is for both the auction and resource allocation agents neural network architectures.

Neural Network	**Description**
Artificial neural networks (McCulloch and Pitts, 1943)	Originally developed as a theoretically approximation for the brain, it was found that for networks with at least one hidden layer could approximate any function (Csáji, 2001). This made neural networks extremely helpful for cases where it would normally be too difficult for a human to specify the exact function. Artificial neural network can be trained through gradient descent to find a close approximation to the true function.

Recurrent neural network (Elman, 1990)	A major weakness of artificial neural networks is that it must use a fixed number of inputs and outputs making it unusable with text, sound or video where previous data is important for understanding the inputs. Recurrent neural network's extend neural networks to allow for connections to previous neurons to "pass on" information. However recurrent neural networks struggle from a vanishing or exploding gradient when training.
Long/Short Term Memory (Hochreiter and Schmidhuber, 1997)	While recurrent neural network's can "remember" previous inputs to the network, it also struggles from the vanishing or exploding gradient problem where gradient tends to zero or infinity making it unusable. This network aims to prevent this by using forget gates that determines how much information the next state will get, allowing for more complexity information to be learnt compared to recurrent neural networks.
Gated Recurrent unit (Chung et al., 2014)	Gated recurrent unit are very similar to long/short term memory, except for the use of a different wiring mechanisms and the use of one less gate, with an update date instead than forgot gates. These changes mean that gated recurrent units allow for them to run faster and are easier to code than long/short term memory, however are not as expressive as LSTMs allowing for less complex functions to be encoded.
Neural Turing Machine (Graves et al., 2014)	Inspired by computers, neural turing machines build on long/short term memory by using an external memory module instead of memory being inbuild to the network. This allows for external observers to understand what is going on much better than other networks due to their black-box nature.
Differentiable neural computer (Graves et al., 2016)	An expansion to the neural turing machine that allows the memory module to scalable in size allowing for additional memory to be added if needed.

TABLE 4.3: Neural network layer descriptions

4.2.1 Justification for Auctioning networks

Outlined in Table 4.3 is the properties of popular neural network layer architectures that would allow for a multiple inputs (except for artificial neural networks). Of the available architecture, long/Short term memory model is the simplest model but still with the complexity to encode the policy. With the neural turing machine and differentiable neural network, these networks are extremely complex and require a large amount of data to train the networks. Also the ability of these networks to be able to store data in external storage is not important as the data doesn't need to be store for future inputs. The opposite problem exists for the recurrent neural network or the gated recurrent unit that they are possibly not complex enough to encode the policy. Because of this, the LSTM network is believed to be the most appropriate network for the proposed agents.

4.2.2 Justification for Resource allocation networks

The justification for the resource allocation agent neural network is very similar justification to the previous subsections 4.2.1. The long/short term memory architecture should be complex enough but it is possible that the ability to use external storage of neural turing machine and differentiable neural network to store the allocation of resource to previous tasks. It is believed that this additional complexity will not allow for the heuristic to do better but it could be investigated in future work.

The reason for the output to be the resource weighting rather than the actual resources is it would require the network to learn a less complex function in comparison. This means that the network learns instead how important the allocation of resources are for a task instead of the exact amount of resources allocated.

Chapter 5

Implementation of the solution

In order to implement a solution from chapter 3, a edge cloud computing environment must be simulated due to the impractically of setting up such a network and in order to train the agents proposed in section 3.3. This chapter splits the implementation into three sections: the environment simulation (section 5.1), define server auction agents and resource allocation agents (section 5.2) and finally training the agents (section 5.3).

The implementation discussed below as written in Python and available to download from Github at `https://github.com/stringtheorys/Online-Flexible-Resource` The reason for the use of python is number of modules available for reinforcement learning and the speed of development.

5.1 Simulating edge cloud computing services

While the aim of the environment is the simulate accurately edge cloud computing server, the implementation of the environment must allow agents to train on interact and training on the environment. Therefore it has been implemented using as an OpenAI gym (Brockman et al., 2016), the de facto standard for implementing reinforcement learning environment for researchers. However the standard specification must be modified due to the problem being multi-agent, multi-step and as a centralised system.

An example for running the environment is in figure **??**. There are three sections to the code; the first is the construct an environment using the constructor, where the environment settings are passed that determines the number of servers and tasks and there attributes. These attributes are determined using uniform random numbers between a maximum and minimum values that are synthesis individually for each variable. The second is to create the environment using the reset function returning the current environment state. The environment state contains a task to be auctioned if any task's auction time step is equal to the environment's current time step as well as a dictionary of server to their current state. Using these states, each server generate's actions either using the auction agent or resource allocation agent depending if if task needs auctioning. The third step is to take a step in environment using the server actions that returns an updated server state, the rewards for the actions, if the environment is finished and an extra information from the steps taken. The rewards are a dictionary of each of the servers with either the winning price for the auctioned task or a list of tasks that have finished either because they ran out of time or completed the task early.

```
# Load the environment with a setting
env = OnlineFlexibleResourceAllocationEnv('settings.env')

# Generate the environment state
server_state = env.reset()

for _ in range(1000):
    # Generate actions
    if server_state.auction_task:
        actions = {
            server: auction_agent.bid(state)
            for server, state in server_state
        }
    else:
        actions = {
            server: resource_allocation_agent.weights(state)
            for server, state in server_state
        }

    # Take environment step
    server_state, reward, done, info = env.step(actions)

    # If the environment is finished then reset it
    if done:
        server_state = env.reset()
\label{listing:example_flexible_resource_env}
```

LISTING 5.1: An example for running the environment

5.1.1 Server resource allocation

A particular complication of the system is to distribute server resources due to the fact that server provide a dictionary of task with a resource weighting that much be converted to the allocated resources for each task. To allocate the compute resources is relatively simple compared to allocating resources for both storage and bandwidth. The algorithm checks first if the weighted resources is greater than the quantity required for the task to finish the compute stage, if this is true then a resources needed for the task to complete the compute stage are allocated. However, this means that the weight resources available for each task is increased due to a task not using all of the resources it could. This means that the algorithm loops till no task can be completed with the weighted resources allocated. Then all of the reminding compute resources are allocated to the remaining tasks.

For allocating storage and bandwidth, the reason for this being more difficult is due to when the server is still loading the task, the server must allocate both storage and bandwidth resources while the bandwidth must be allocated to the task sending results. Because of this, a tension exists between these two operation for allocating resources. The implemented algorithm, gives priority for allocating resources to the tasks sending results as these tasks are more likely to be finished and will not penalise the server for not completing the task within the deadline. To allocate resources, a similar function to used to the one for allocating compute resources. First a check if done using the weighted bandwidth resources to see if any task sending results will be finished with the resources, if so these resources are allocated. This process if repeat for the server loading any task with the additional check that there is enough available storage for the bandwidth resources being allocated. For any remaining task, this process if repeated till all of the available resources are allocated in the time step. As a results, using this algorithm, the converting between weightings to resources allowing for allocating of almost all of the server's resources with no resources unused.

5.2 Implementing Auction and resource allocation agents

Each server has to have a unique policy for bidding on tasks and allocating resources that are proposed in section 3.3. These policies are referred to as the auction agents, for bidding on tasks and the resource weighting agent, for allocating of resources. However determining these policies is a difficult due to the complexity and the interaction of task attributes and the server resources. Because of this, the ability of reinforcement learning to interact an environment and learn a policy in order to maximise its reward over time is integral to this project. Therefore a range of different reinforcement learning techniques have been implemented, outlined in Table 5.1, in order to explore the different options that a server would have available to learn its policies.

5.2.1 Implementing reinforcement learning policies

These policies were implemented using tensorflow Abadi et al. (2015), a python module developed by Google that provides programmers the ability to construct neural network and backpropagation with a loss function. A particular problem with this project is the use of recurrent neural network and inputs not having a fixed length. This causes an issue with tensorflow due to the requirement for tensor, the base for tensorflow operation, to have a fixed size. To allow for efficient using of backpropagation with multiple inputs being computed at the same time in order to compute a minibatch not just stochastic gradient descent. To do requires the use of the tensorflow preprocessing module to pad all of the input to be the same size.

5.2.2 Rewards functions

As explained in the background review for reinforcement learning (section 2.2), the q values is an approximation for the estimated discounted reward in the future given an action. Therefore the rewards that an agent receives for taking an action is extremely important to enable the agent to learn. This problem of complex reward functions are a known problem for DQN agent to deal with (Mnih et al., 2013) while policy gradients can deal with this due to learning the policy rather than the q values (Sutton and Barto, 2018).

Policy Type	Explanation
Dqn (Mnih et al., 2015)	A standard deep Q learning agent that discretizes the action space
Double Dueling DQN (van Hasselt et al., 2015; Wang et al., 2015)	A combination of two heuristics for the standard deep Q learning agents that uses a modified td target function and a modified networks that separates state value and action advantage.
Categorical Dqn (Bellemare et al., 2017)	Standard deep Q learning agents return a scalar value (representing the q value) for each action. Distributional Dqn changes the output value to a probability distribution over action values that is helpful due to the stochastic nature from the problem (from the agents perspective).
Deep deterministic policy gradient (Silver et al., 2014)	As the action space is continuous, DDPG allows for investigation of the difference between continuous and discrete action spaces of the DQN agents and policy gradient can be more effective at learning a policy where the reward function is too complex for DQN to model.
Twin delay DDPG (Fujimoto et al., 2018)	Like the Double Dueling DQN agents, TD3 includes a couple new heuristics, like a twin critic to prevent the actor tricking the critic and delaying the updates for actors compared to critic.
D4PG (Barth-Maron et al., 2018)	Like the categorical dqn, d4pg add a heuristic for the critic to output a value probability distribution that allows for better approximation for environment that are stochastic in nature.

TABLE 5.1: Table of implemented reinforcement learning algorithms

For the auction, the reward is based on the winning price of the task at the time step when the auction is won. If the task fails, the reward is instead multiplied by a negative constant (referred to as ”failed_reward_multiplier”) in order to discourage the auction agent from bidding on tasks that it wouldn’t be able to complete. Because of this, the price of zero is treated as a non-bid, resulting in the agent getting a reward of zero and not being considered in the auction. The final possible reward for an auction agent is that the agent fails to win the task in the auction by bidding too high, as a result, the reward is constant called ”failed_auction_reward”. The constant’s default value is -0.05 as a way of encouraging the agent to change there bid.

For resource allocation, the reward function is much simpler than the auction reward function as it only needs to consider the task being weighted at the time and rewards from other tasks allocated at the time. This is as, a task must consider its actions in consider with the resource requirements of other allocated tasks. For successfully finishing a task, the reward is 1 while the reward for if

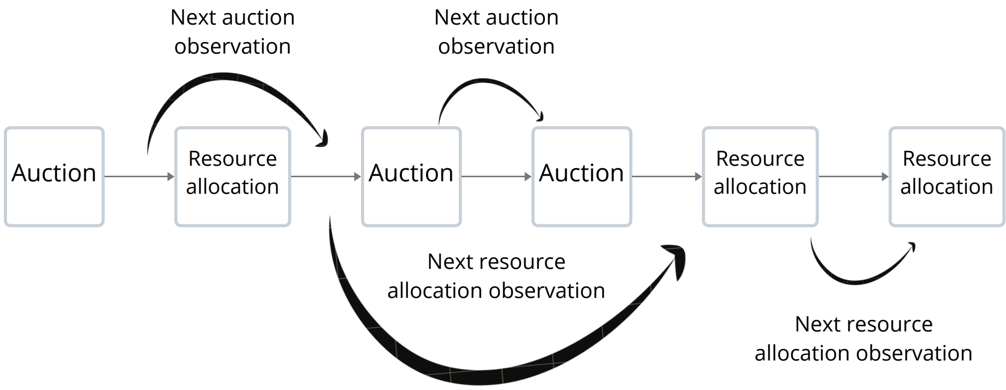


FIGURE 5.1: Environment server agents observations

the task has failed is -1.5 to make failing a task more costly than completing a task. But when a task's action is not under consideration, this reward is multiplied by 0.4 as while this rewards impact the task, their value is not as important to the task current consideration. While these rewards don't consider the price payed for the task instead valuing each task equally with the aim of forcing the task allocate resources to finish all tasks not just the valuable ones. Using this information, the reward function simply just the sum of finished tasks rewards in the next time step.

5.2.3 Agent observations

Both deep q networks and policy gradient algorithm are based on the Q function (explained in section 2.2)) which tries to approximate the reward at the next time step. To do this required a reward function and a next observation to compare to in order to train these agents. This can be described as a tree with the current state being the root and all lefts being the next states however for this problem, this formulation has several problems due to there being two environment happening at the same time meaning that the next state is not clear as shown in figure 5.1.

For the resource allocation agent, a trick is implemented such that the next observation for the agent is not the actual next resource allocation observation as shown in the figure but a generated observation from the resulting server state due to agent's actions. This is identical to the last case in the figure where no auction occurs between resource allocation steps. Because of this, the resource allocation q value is able to approximate the reward for the results of its actions directly and makes it appear to the agent that there is no auction steps with the observations and next observations for training.

For the auction agent, as the agent observations require an auction task to compare to and select an action (the task bidding price), a trick like the one implemented for the resource allocation agent is not possible to implement. Therefore during each server's last observation to recorded such that when the next auction occurs, this new task observation can be used as the server's next observation. This is a suboptimal solution as the agent as the next task been auctioned in completely random as well as between the current task observation and the next auction observation, an unknown number of resource allocation steps could have been taken meaning that the resource demand by the other currently allocated tasks has changed. A possible solution that has not been implement in this project is N step prediction (Sutton, 1988) where the agent doesn't predict the q value of the next environment, but the q value of the environment in N steps time. This is believed to help reduce the amount of randomness in the server's observation to train on.

5.3 Training agents

The first section of this chapter simulates edge cloud computing servers (section 5.1) while the second section implements auction and resource allocation agents that can interact with such environment and use a range of algorithms to learn from. By combining both of these together allows agents to continually interact with environments in order to learn over time their respective policies.

Neural networks, the bases of the reinforcement learning agents implemented, often require huge amounts of data and high spec GPUs to run efficiently. Because of this, Iridis 5, University of Southampton supercomputer was utilised with GTX1050 GPUs to train these agents for long periods of time and on mass. During training, for each episode a random environment was generated from a list of possible settings in which the agents would be allocated to random servers. The environment was run till the end, with the agent observation being added to their replay buffers after each actions that were chosen epsilon greedily.

But after every 5 episodes, the agents would be evaluated using a set of environment that were pre-generated and saved at the beginning of training. This allows the same environments over training in order to have a constant metric in order to compare the agents over time. The actions taken are recorded to be used in evaluation (chapter 7) with the number of completed and failed tasks being stored about the resource allocation agent and the winning prices being stored about the auction agents. Plus an action histograms for each agents in order to view how agents bid and weightings are distributed.

5.3.1 Agent training hyperparameters

Agent	Properties name	Value	Explanation
Task Pricing	limit_parallel_tasks	None	
RL Agent	batch_size	32	
RL Agent	error_loss_fn	tf.losses.huber_loss	
RL Agent	initial_training_replay_size	5000	
RL Agent	training_freq	2	
RL Agent	discount_factor	0.9	
RL Agent	replay_buffer_length	25000	
RL Agent	save_frequency	25000	
RL Agent	training_loss_log_freq	250	
Task Pricing RL Agent	reward_scaling	1	
Task Pricing RL Agent	failed_auction_reward	-0.05	
Task Pricing RL Agent	failed_multiplier	-1.5	
Resource weighting RL Agent	other_task_discount: float = 0.4		
Resource weighting RL Agent	success_reward	1	
Resource weighting RL Agent	failed_reward	-1.5	
Dqn Agent	optimiser	tf.keras.optimizers.Adam()	
Dqn Agent	target_update_tau	1.0	
Dqn Agent	target_update_frequency	2500	
Dqn Agent	initial_epsilon	1	
Dqn Agent	final_epsilon	0.1	

Dqn Agent	epsilon_steps	10000	
Dqn Agent	epsilon_update_frequency	100	
Dueling Dqn Agent	double_loss	True	
Categorical Dqn Agent	max_value	-20.0	
Categorical Dqn Agent	min_value	25.0	
Categorical Dqn Agent	num_atoms	21	
Ddpg Agent	actor_optimiser	tf.keras.optimizers.RMSprop(lr=0.0001)	
Ddpg Agent	critic_optimiser	tf.keras.optimizers.RMSprop(lr=0.0005)	
Ddpg Agent	initial_epsilon_std	0.8	
Ddpg Agent	final_epsilon_std	0.05	
Ddpg Agent	epsilon_steps	20000	
Ddpg Agent	epsilon_update_frequency	100	
Ddpg Agent	target_update_tau	1.0	
Ddpg Agent	actor_target_update_frequency	3000	
Ddpg Agent	critic_target_update_frequency	1500	
Ddpg Agent	upper_action_bound	30.0	
Task pricing Ddpg Agent	min_value	-100.0	
Task pricing Ddpg Agent	max_value	100.0	
Resource allocation Ddpg Agent	min_value	-20	
Resource allocation Ddpg Agent	max_value	15	
TD3 Agent	twin_critic_optimiser	tf.keras.optimizers.Adam()	
TD3 Agent	actor_update_frequency	3	

TABLE 5.2: Agent hyperparameters

Chapter 6

Testing

To compare the implemented agent policies and neural networks architectures from chapter 5, tests are implemented to evaluation the effectiveness of training for each of these agents and In order to compare the effectiveness of training techniques, a range of metrics are used for each agent. For the auction agent the metrics are a histogram winning prices, number of no bids, number of failed tasks, number of completed tasks and a histogram of actions taken. For the resource allocation agents the metrics are a histogram of weighting, number of failed tasks and number of completed tasks. These training tests fall into several distinct sections that are explained in Table 6.1.

Test Name	Explanation
Multi vs Single environment training settings	There are huge ranges of possible environment settings that agents could be trained for. This test investigates how good single environment trained agents react to a new environment and how agents compare on an environment when an agent is only trained on that environment or on multiple environments.
Multi vs Single agent environment training settings	As the Vickrey auction is incentive compatible then it is possible to train both the auction agents and resource allocation against itself with a single agent. This test investigates the difference between the agent policies when self-trained or train with other agents.
Reinforcement Learning Policy testing	As multiple different reinforcement learning policy outlined in table 5.1, this test compares the results of the different agents against each other.
Neural network architecture testing	There are a wide-range of compatible neural network architectures that agents can use, as outlined in table 4.3. To use these agents, the underlying policies are kept the same with a range of model networks are trained.
Fixed heuristic policies testing	In order to compare to the reinforcement learning policies, a number of fixed heuristic policies are investigated both against other heuristics and reinforcement learning policies.

TABLE 6.1: Table of Testing Areas

Chapter 7

Evaluation of the implementation

Chapter 8

Conclusion and future work

Appendices

Appendix A: Paper

This paper was been produced with the authors being myself, Dr Sebastian Stein, Professor Tim Norman from Southampton University, Dr Fidan Mehmeti, Professor Tom La Porta, Caroline Rubein from Pennsylvania State University and Dr Geeth Demel from IBM and within this project is referred to as Towers et al. (2020).

Auction-based Mechanisms for Allocating Elastic Resources in Edge Clouds

Paper #1263

ABSTRACT

Edge clouds enable computational tasks to be completed at the edge of the network, without relying on access to remote data centres. A key challenge in these settings is the limited computational resources that need to be allocated to many self-interested users. Here, existing resource allocation approaches usually assume that tasks have inelastic resource requirements (i.e., a fixed amount of compute time, bandwidth and storage), that may result in inefficient resource use due to unbalanced requirements. To address this, we propose a novel approach that takes advantage of the elastic nature of some of the resources, e.g., to trade-off computation speed with bandwidth allowing a server to execute more tasks by their deadlines. We describe this problem formally, show that it is NP-hard and then propose a scalable approximation algorithm. To deal with the self-interested nature of users, we show how to design a centralized auction that incentives truthful reporting of task requirements and values. Moreover, we propose novel auction-based decentralized approaches that are not always truthful, but that limit the information required from users and that can be adjusted to trade off convergence speed with solution quality. In extensive simulations, we show that considering the elasticity of resources leads to a gain in utility of around 20% compared to existing fixed approaches and that our novel auction-based approaches typically achieve 95% of the theoretical optimal.

KEYWORDS

Edge clouds; elastic resources; auctions

1 INTRODUCTION

In the last few years, cloud computing [2] has become a popular solution to run data-intensive applications remotely. However, in some application domains, it is not feasible to rely a remote cloud, for example when running highly delay-sensitive and computationally-intensive tasks, or when connectivity to the cloud is intermittent. To deal with such domains, *mobile edge computing* [13] has emerged as a complementary paradigm, where computational tasks are executed at the edge of mobile networks at small data-centers, known as *edge clouds*.

Mobile edge computing is a key enabling technology for the Internet-of-Things (IoT) [6] and in particular applications in smart cities [19] and disaster response scenarios [9]. In these applications, low-powered devices generate computational tasks and data that have to be processed quickly on local edge cloud servers. More specifically, in smart cities, these devices could be smart intersections that collect data from road-side sensors and vehicles to produce an efficient traffic light sequence to minimize waiting times [14]; or it could be CCTV cameras that analyse video feeds for suspicious behaviour, e.g., to detect a stabbing or other crime in progress [20]. In disaster response, sensor data from autonomous vehicles (including video, sonar and LIDAR) can be aggregated in real time to produce maps of a devastated area, search for potential victims and help first responders in focusing their efforts to save lives [1].

To accomplish these tasks, there are typically several types of resources that are needed, including communication bandwidth, computational power and data storage resources [7], and tasks are generally delay-sensitive, i.e., have a specific completion deadline. When accomplished, different tasks carry different values for their owners (e.g., the users of IoT devices or other stakeholders such as the police or traffic authority). This value will depend on the importance of the task, e.g., analysing current levels of air pollution may be less important than preventing a large-scale traffic jam at peak times or tracking a terrorist on the run. Given that edge clouds are often highly constrained in their resources [12], we are interested in allocating tasks to edge cloud servers to maximize the overall social welfare achieved (i.e., the sum of completed task values). This is particularly challenging, because users in edge clouds are typically self-interested and may behave strategically [3] or may prefer not to reveal private information about their values to a central allocation mechanism [18].

An important shortcoming of existing work of resource allocation in edge clouds, e.g., [3, 7], is that it assumes tasks have strict resource requirements — that is, each task consumes a fixed amount of computation (CPU cycles per time), takes up a fixed amount of bandwidth to transfer data and uses up a fixed amount of storage on the server. However, in practice, edge cloud servers have some flexibility in how they allocate limited resources to each task. In more detail, to execute a task, the corresponding data and/or code first has to be transferred to the server it is assigned to, requiring some bandwidth. This then takes up storage on the server. Next, the task needs computing power from the server in terms of CPU cycles per time. Once computation is complete, the results have to be transferred back to the user, requiring further bandwidth. Now, while the the storage capacity at the server for every task is *strict*, since the task cannot be run unless all the data is stored, the bandwidth and compute speed allocated to the task can be *elastic*. This allows flexibility in the resource allocation process enabling resources to be shared evenly, prevent resource self-interested users and for more task to receive service simultaneously.

Against this background, we make the following novel contributions to the state of the art:

- We formulate an optimization problem for assigning the tasks to the servers, whose objective is to maximize total

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 2020, Auckland, New Zealand

https://doi.org/doi

social welfare, taking into account resource limitations and elastic allocation of resources.

- We prove that the problem is NP-hard and propose an approximation algorithm with a performance guarantee of $\frac{1}{n}$, where n is the number of tasks, and a linearithmic computational complexity, i.e., $O(n \log(n))$.
- We propose a range of auction-based mechanisms to deal with the self-interested nature of users. These offer various trade-offs regarding truthfulness, optimality, scalability, information requirements from users, communication overheads and decentralization.
- Using extensive realistic simulations, we compare the performance of our algorithm against other benchmark algorithms, and show that our algorithm outperforms all of them, while at the same time being within 95% to the optimal solution.

The paper is organized as follows. In the next section we discuss related work. This is followed by the problem formulation in Section 3. Our novel resource allocation mechanisms are presented in Section 4. In Section 5, we evaluate the performance of our mechanisms and compare them against the optimal solution and other benchmarks. Finally, Section 6 concludes the work.

2 RELATED WORK

There is a considerable amount of research in the area of resource allocation and pricing in cloud computing, some of which use auction mechanisms to deal with competition [3, 4, 11, 22]. However, these approaches assume that users request a fixed amount of resources system resources and processing rates, with the cloud provider having no control over the speeds, only the servers that the task was allocated to. In our work, tasks' owners report deadlines and overall data and computation requirements, allowing the edge cloud server to distribute its resources more efficiently based on each task's requirements.

Our problem is related to multidimensional knapsack problems. In particular, Nip et al. [15] consider flexibility in the allocation, with linear constraints that are used for elastic weights. The paper provides a pseudo-polynomial time complexity algorithm for solving this problem to maximize the values in the knapsack. Our problem case is similar to their problem, but our problem has non-linear constraints due to the deadline constraint, so their algorithm cannot be applied here.

Other closely related work on resource allocation in edge clouds [7] considers both the placement of code/data needed to run a specific task, as well as the scheduling of tasks to different edge clouds. The goal there is to maximize the expected rate of successfully accomplished tasks over time. Our work is different both in the setup and the objective function. Our objective is to maximize the value over all tasks. In terms of the setup, they assume that data/code can be shared and they do not consider the elasticity of resources.

3 PROBLEM FORMULATION

In this section we first describe the system model. Then, we present the optimization problem and prove its NP-hardness.

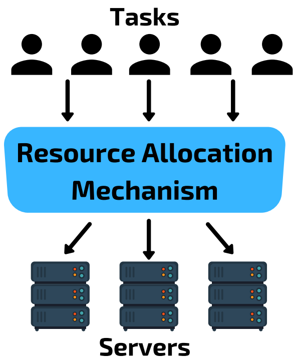


Figure 1: System Model

3.1 System model

A sketch of the system is shown in Fig. 1. We assume that in the system there is a set of servers $I = \{1, 2, \ldots, |I|\}$ servers, which could be edge clouds that can be accessed either through cellular base stations or WiFi access points (APs). Servers have different types of resources: storage for the code/data needed to run a task (e.g., measured in GB), computation capacity in terms of CPU cycles per time interval (e.g., measured in FLOP/s), and communication bandwidth to receive the data and to send back the results of the task after execution (e.g., measured in Mbit/s). We assume that the servers are heterogeneous in all their characteristics. More formally, we denote the storage capacity of server i with S_i, computation capacity with W_i, and the communication capacity with R_i.

There is a set $J = \{1, 2, \ldots, |J|\}$ of different tasks that require service from one of the servers.[1] Every task $j \in J$ has a value v_j that represents the value of running the task to its owner. To run any of these tasks on a server requires storing the appropriate code/data on the same server. These could be, for example, a set of images, videos or CNN layers in identification tasks. The storage size of task j is denoted as s_j with the rate at which the program is transferred to the server being $s_j^{'}$. For a task to be computed successfully, it must fetch and execute instructions on a CPU. We consider the total number of CPU cycles required for the program to be w_j, where the rate at which the CPU cycles are assigned to the task per unit of time is $w_j^{'}$. Finally, after the task is run and the results obtained, the latter need to be sent back to the user. The size of the results for task j is denoted with r_j, and the rate at which they are sent back to the user is $r_j^{'}$. Every task has its deadline, denoted by d_j. This is the maximum time for the task to be completed in order for the user to derive its value. This time includes: the time required to send the data/code to the server, run it on the server, and get back the results. We assume that there is an *all* or *nothing* task execution reward scheme, meaning that for the task value to be awarded the entire task must be run and the results sent back within the deadline.

[1]We focus on a single-shot setting in this paper. In practice, an allocation mechanism would repeat the allocation decisions described here over regular time intervals, with longer-running tasks re-appearing on consecutive time intervals. We leave a detailed study of this to future work.

3.2 Optimization problem

Given the aforementioned assumptions, the optimal assignment of tasks to servers and optimal allocation of resources in a server to the tasks assigned to that server is obtained as a solution to the following optimization problem. Here, the decision variables are $x_{i,j} \in \{0,1\}$ (whether to run task j on server i) as well as $s_j^{'}$, $r_j^{'}$ and $w_j^{'}$ (indicating the bandwidth rates for transferring the code, for returning the results and the CPU cycles per unit of time, respectively).

$$\max \sum_{\forall j \in J} v_j \left(\sum_{\forall i \in I} x_{i,j} \right) \tag{1}$$

s.t.

$$\sum_{\forall j \in J} s_j x_{i,j} \leq S_i, \qquad \forall i \in I, \tag{2}$$

$$\sum_{\forall j \in J} w_j^{'} x_{i,j} \leq W_i, \qquad \forall i \in I, \tag{3}$$

$$\sum_{\forall j \in J} (r_j^{'} + s_j^{'}) \cdot x_{i,j} \leq R_i, \qquad \forall i \in I, \tag{4}$$

$$\frac{s_j}{s_j^{'}} + \frac{w_j}{w_j^{'}} + \frac{r_j}{r_j^{'}} \leq d_j, \qquad \forall j \in J, \tag{5}$$

$$0 \leq s_j^{'} \leq \infty, \qquad \forall j \in J, \tag{6}$$

$$0 \leq w_j^{'} \leq \infty, \qquad \forall j \in J, \tag{7}$$

$$0 \leq r_j^{'} \leq \infty, \qquad \forall j \in J, \tag{8}$$

$$\sum_{\forall i \in I} x_{i,j} \leq 1, \qquad \forall j \in J, \tag{9}$$

$$x_{i,j} \in \{0,1\}, \qquad \forall i \in I, \forall j \in J. \tag{10}$$

The objective (Eq.(1)) is to maximize the total value over all tasks (i.e., the social welfare). Task j will receive the full value v_j only if it is executed entirely and the results are obtained within the deadline for that task. Constraint (Eq.(2)) relates to the finite storage capacity of every server to store code/data for the tasks that are to be run. The finite computation capacity of every server is expressed through Eq.(3), whereas Eq.(4) denotes the constraint on the communication capacity of the servers. As can be seen, the communication bandwidth comprises two parts: one part to send the data/code or request to the server, and the other part to get the results back to the user.[2] Constraint Eq.(5) is the deadline associated with every task, where the total time of the task in the system is the sum of the time to send the request and code/data to the server, time to run the task, and the time it takes the server to send all the results to the user. Note that if a task is not run on any server, this constraint can be satisfied by choosing arbitrarily high bandwidth and CPU rates (without being constrained by the resource limits of any server). The rates at which the code is sent, run and the results are sent back are all positive and finite (Eqs. (6), (7), (8)). Further, every task is served by at most one server (Eq.(9)). Finally, a task is either served or not (Eq.(10)).

[2]Not that sending and receiving data will not always overlap, but for tractability we assume they deplete a common limited bandwidth resource per time step. This ensures that the bandwidth constraint is always satisfied in practice.

Complexity: In the following we show that this optimization problem is NP-hard.

Theorem 3.1. *The optimization problem* (1)-(10) *is NP-hard.*

Proof. The optimization problem without constraint (5) is a 0-1 multidimensional knapsack problem [10], which is a generalization of a simple 0-1 knapsack problem. The latter is an NP-hard problem [10]. Given this, it follows that the 0-1 multidimensional knapsack problem is also NP-hard. Since optimization problem (1)-(10) is a generalization of a 0-1 multidimensional knapsack problem, it follows that it is NP-hard as well. □

Before we propose our novel allocation mechanisms for the allocation problem with elastic resources, we briefly outline an example that illustrates why considering this elasticity is important. In this example, there are 12 potential tasks and 3 servers (the exact settings can be found in table 2 for the tasks and table 1 for the servers).

Figure 2 shows the best possible allocation if tasks have fixed resource requirements. The resource speeds were chosen such to the minimum total resource usage that the task would require from the deadline. Here, 9 of the tasks are run, resulting in a total social welfare of 980 due to the limitation of the server's computation and the task requirement not being balanced.

In contrast to this, Figure 3 depicts the optimal allocation if elastic resources are considered. Here, it is evident that all of the resources are used by the servers whereas the fixed (in figure 2) cant do this. In total, the elastic approach manages to schedule all 12 tasks within the resource constraints, achieving a total social welfare of 1200 (an 19% improvement over the fixed approach).

The figures represent resource usage of the servers by the three bars relating to each of this resources (storage, CPU and bandwidth). For each task that is allocated to the server, the percentage of the resource's used is bar size. Then, for the tasks that are assigned to corresponding servers, the percentage of used resources are also depicted.

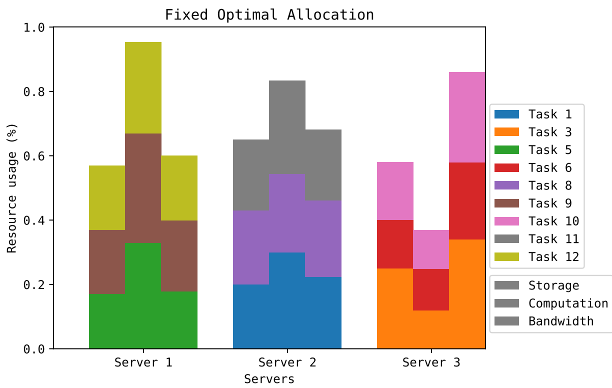


Figure 2: Optimal solution with fixed resources. Due to not being able to balance out the resources, bottlenecks on the server 1 and 2's computation have occurred

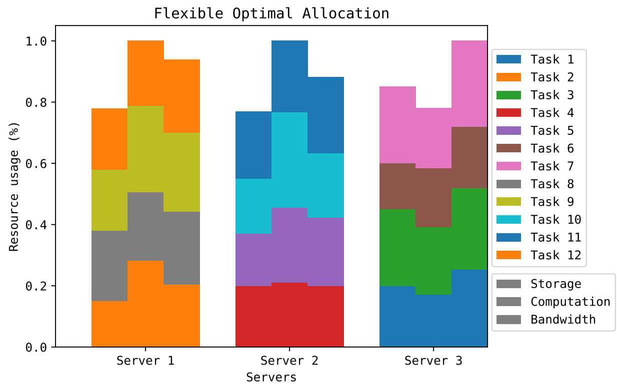


Figure 3: Optimal solution with elastic resources. Compared to the fixed allocation, the elastic allocation is able to fully use all of its resources

Name	S_i	W_i	R_i
Server 1	400	100	220
Server 2	450	100	210
Server 3	375	90	250

Table 1: Servers - Table of server attributes

Name	v_j	s_j	w_j	r_j	d_j	s_j'	w_j'	r_j'
Task 1	100	100	100	50	10	30	27	17
Task 2	90	75	125	40	10	22	32	15
Task 3	110	125	110	45	10	34	30	17
Task 4	75	100	75	35	10	27	21	13
Task 5	125	85	90	55	10	24	28	17
Task 6	100	75	120	40	10	20	32	16
Task 7	80	125	100	50	10	31	30	19
Task 8	110	115	75	55	10	30	22	20
Task 9	120	100	110	60	10	27	29	24
Task 10	90	90	120	40	10	25	30	17
Task 11	100	110	90	45	10	30	26	16
Task 12	100	100	80	55	10	24	24	22

Table 2: Tasks - Table of task attributes, the columns for resource speeds (s_j', w_j', r_j') is for fixed speeds which the flexible allocation does not take into account. The fixed speeds is the minimum required resources to complete the task within the deadline constraint.

4 FLEXIBLE RESOURCE ALLOCATION MECHANISMS

In this section, we propose several mechanisms for solving the resource allocation problem with elastic resources. First, we discuss a centralized greedy algorithm (detailed in Section 4.1) with a $\frac{1}{|J|}$ performance guarantee and polynomial run-time. Then, we consider settings where task users are self-interested and may either report their task values and requirements strategically or may wish to limit the information they reveal to the mechanism. To deal with such cases, we propose two auction-based mechanisms, one of which can be executed in a decentralized manner (in Sections 4.2 and 4.3).

4.1 Greedy Mechanism

As solving the allocation problem with elastic resources is NP-hard, we here propose a greedy algorithm (Algorithm 1) that considers tasks individually, based on an appropriate prioritisation function.

More specifically, the greedy algorithm does this in two stages; the first sorts the tasks and the second allocates them to servers. A value density function is applied to each of the task based on its attributes: value, required resources and deadlines. Stage one uses this function to sort the list of tasks. The second stage then iterates through the tasks in the given order, applying two heuristics to each task: one to select the server and another to allocate resources. The first of these heuristics, called the server selection heuristic, works by checking if a server could run the task if all of its resources were to be used for meeting the deadline constraint (eq 5) then calculating how good it would be for the job to be allocated to the server. The second heuristic, called the resource allocation heuristic, finds the best permutations of resources to minimise a formula, i.e., the total percentage of server resources used by the task.

In this paper we prove that the lower bound of the algorithm is $\frac{1}{|J|}$ (where $|J|$ is the number of jobs) using the value of a task as the value density function and using any feasible server selection and resource allocation heuristic. However we found that the task value heuristic is not the best heuristic as it does not consider the effect of the deadline or resources used for a job. In practice, the following heuristic often works better: $\frac{v_j \cdot (s_j + w_j + r_j)}{d_j}$. For the server selection heuristic we use $argmin_{\forall i \in I} S_i' + W_i' + R_i'$, where S_i', W_i', R_i' are the server's available storage, computation and bandwidth resources respectively. While for the resource allocation heuristic we use $min \frac{W_i'}{w_j'} + \frac{R_i'}{s_j' + r_j'}$.

Theorem 4.1. *The lower bound of the greedy mechanism is $\frac{1}{n}$ of the optimal social welfare*

Proof. Taking the value of a task as the value density function, the first task allocated will have a value of at least $\frac{1}{n}$ total values of all jobs. As the allocation of resources for a task is not optimal, allocation of subsequent tasks is not guaranteed. Therefore, as the optimal social welfare must be the total values of all jobs or lower then the lower bound of the mechanism must be $\frac{1}{n}$ of the optimal social welfare. □

In figure 4, an example allocation using the algorithm is shown using the model from tables 1 and 2. The algorithm uses the recommend heuristic proposed above and allows for all tasks to be allocated achieving 100% of the flexible optimal in figure 3.

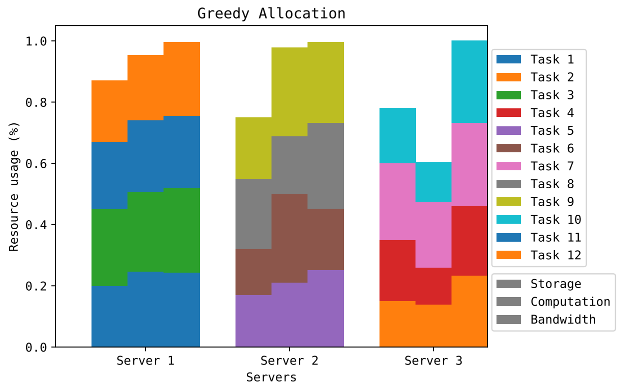


Figure 4: Example Greedy allocation using model from table 2 and 1

Algorithm 1 Greedy Mechanism

Require: J is the set of tasks and I is the set of servers
Require: $S_i^{'}$, $W_i^{'}$ and $R_i^{'}$ is the available resources (storage, computation and bandwidth respectively) for server i.
Require: $\alpha(j)$ is the value density function of a task
Require: $\beta(j, I)$ is the server selection function of a task and set of servers returning the best server, or $\emptyset$ if the task is not able to be run on any server
Require: $\gamma(j, i)$ is the resource allocation function of a task and server returning the loading, compute and sending speeds
Require: $sort(X, f)$ is a function that returns a sorted list of elements in descending order, based on a set of elements and a function for comparing elements
  $J^{'} \leftarrow sort(J, \alpha)$
  for all $j \in J^{'}$ **do**
    $i \leftarrow \beta(j, I)$
    if $i \neq \emptyset$ **then**
      $s_j^{'}, w_j^{'}, r_j^{'} \leftarrow \gamma(j, i)$
      $x_{i,j} \leftarrow 1$
    end if
  end for

Theorem 4.2. *The time complexity of the greedy algorithm is $O(|J|\,|I|)$, where $|J|$ is the number of tasks and $|I|$ is the number of servers. Assuming that the value density and resource allocation heuristics have constant time complexity and the server selection function is $O(|I|)$.*

Proof. The time complexity of the stage 1 of the mechanism is $O(|J| \log(|J|))$ due to sorting the tasks and stage 2 has complexity $O(|J|\,|I|)$ due to looping over all of the tasks and applying the server selection and resource allocation heuristics. Therefore the overall time complexity is $O(|J|\,|I| + |J| \log(|J|) = O(|J|\,|I|)$. □

4.2 Critical Value Auction

Due to the problem case being non-cooperative, if the greedy mechanism was used to allocate resources such that the value is the price paid. This is open to manipulation and misreporting of task attributes like the value, deadline or resource requirements. Therefore in this section we propose an auction that is weakly-dominant for tasks to truthfully report it attributes.

Single-Parameter domain auctions are extensively studied in mechanism design [16] and are used where an agent's valuation function can be represented as single value. The task price is calculated by finding the task's value such that if the value were any smaller, the task could not be allocated. This value is called the critical value. This has been shown to be a strategyproof [17] (weakly-dominant incentive compatible) auction so it is a weakly-dominant strategy for a task to honestly reveal its value.

The auction is implemented using the greedy mechanism from section 4.1 to find an allocation of tasks using the reported value. Then for each task allocated, the last position in the ordered the task list such that the task would still allocated is found. The critical value of the task is then equal to the inverse of the value density function where the density is the density of the next task in the list after that position.

In order that the auction is strategyproof, the value density function is required to be monotonic so that misreporting of any task attributes will result in the value density decreasing. Therefore a value density function of the form $\frac{v_j d_j}{\alpha(s_j, w_j, r_j)}$ must be used so that the auction is strategyproof.

Theorem 4.3. *The value density function $\frac{v_j d_j}{\alpha(s_j, w_j, r_j)}$ is monotonic for task j assuming the function $\alpha(s_j, w_j, r_j)$ is monotonic decreasing.*

Proof. In order to misreport the task private value and deadline must be less than the true value. The opposite is true for the required resources (storage, compute and result data) with the misreported value being greater than the true value. Therefore the α function will increase as the resource requirements increase as well, meaning that density will decrease. □

4.3 Decentralised Iterative Auction

VCG (Vickrey-Clark-Grove) auction [21] [5] [8] is proven to be economically efficient, budget balanced and incentive compatible. A task's price is found by the difference of the social welfare for when the task exists compared to the social welfare when the task doesn't exist. Our auction uses the same principle for pricing by finding the difference between the current server revenue and the revenue when the task is allocated (at £0).

The auction iteratively lets a task advertise its requirements to all of the servers who respond with their price for the task. This price is equal to the server's current revenue minus the solution to the the problem in section 4.3.1 plus a small value called the price change variable. Being the reverse of the VCG mechanism, such that the price is found for when the task exists rather than when it doesn't exist. The price change variable allows for the increase in the revenue of the server and is can be chosen by the server. Once all of the server have responded, the task can compare the minimum server price to its private value. If the price is less then the task will accept the servers with the minimum price offer, otherwise the task will stop looking as the price for the task to run on any server is greater than its reserve price.

To find the optimal revenue for a server m given a new task p and set of currently allocated tasks N has a similar formulation to section 3.2. With an additional variable is considered, a task's price being p_n for task n.

4.3.1 Server problem case.

$$\max \sum_{\forall n \in N} p_n x_n \tag{11}$$

$$\text{s.t.} \tag{12}$$

$$\sum_{\forall n \in N} s_n x_n + s_p \leq S_m, \tag{13}$$

$$\sum_{\forall n \in N} w_n^{'} x_n + w_p \leq W_m, \tag{14}$$

$$\sum_{\forall n \in N} (r_n^{'} + s_n^{'}) \cdot x_n + (r_p^{'} + s_p^{'}) \leq R_m, \tag{15}$$

$$\frac{s_n}{s_n^{'}} + \frac{w_n}{w_n^{'}} + \frac{r_n}{r_n^{'}} \leq d_n, \qquad \forall n \in N \cup \{p\}, \tag{16}$$

$$0 \leq s_n^{'} \leq \infty, \qquad \forall n \in N \cup \{p\} \tag{17}$$

$$0 \leq w_n^{'} \leq \infty, \qquad \forall n \in N \cup \{p\} \tag{18}$$

$$0 \leq r_n^{'} \leq \infty, \qquad \forall n \in N \cup \{p\} \tag{19}$$

$$x_n \in \{0, 1\}, \qquad \forall n \in N \tag{20}$$

The objective (Eq.(11)) is to maximize the price of all tasks (not including the new task as the price is zero). The server resource capacity constraints are similar to the constraints in the standard model set out in section 3.2 however with the assumption that the task k is running so there is no need to consider if the task is running or not. The deadline and non-negative resource speeds constraints (5, 6, 7 and 8) are all the same equation with the new task included with all of the other tasks. The equation to check that a task is only allocated to a single server is not included as only server i considers the task k's price.

In auction theory, four properties are considered: Incentive compatible, budget balanced, economically efficient and individual rationality.

- Budget balanced - Since the auction is run without an auctioneer, this allows for the auction to be run in a decentralised way resulting in no "middlemen" taking some money so all revenue goes straight to the servers from the tasks
- Individually Rational - As the server need to confirm with the task if it is willing to pay an amount to be allocated, the task can check this against its secret reserved price preventing the task from ever paying more than it is willing
- Incentive Compatible - Misreporting can give a task as if the task can predict the allocation of resources from server to tasks then tasks can misreport so to be allocate to a certain server that otherwise would result in the task being unallocated.
- Economic efficiency - At the begin then task are almost randomly assigned in till server become full and require kicking tasks off, this means that allocation can fall into a local price maxima meaning that the server will sometime not be 100% economically efficient.

Algorithm 2 Decentralised Iterative Auction

Require: I is the set of servers
Require: J is the set of unallocated tasks, which initial is the set of all tasks to be allocated
Require: $P(i, k)$ is solution to the problem in section 4.3.1 using the server i and new task k. The server's current tasks is known to itself and its current revenue from tasks so not passed as arguments.
Require: $R(i, k)$ is a function returning the list of tasks not able to run if task k is allocated to server i
Require: $\leftarrow_R$ will randomly select an element from a set
  while $|J| > 0$ **do**
    $j \leftarrow_R J$
    $p, i \leftarrow argmin_{i \in I} P(i, j)$
    if $p \leq v_j$ **then**
      $p_j \leftarrow p$
      $x_{i,j} \leftarrow 1$
      for all $j^{'} \in R(i, j)$ **do**
        $x_{i,j^{'}} \leftarrow 0$
        $p_j^{'} \leftarrow 0$
        $J \leftarrow J \cup j^{'}$
      end for
    end if
    $J \leftarrow J \setminus \{j\}$
  end while

The algorithm 2 is a centralised version of the decentralised iterative auction. It works through iteratively checking a currently unallocated job to find the price if the job was currently allocated on a server. This is done through first solving the program in section 4.3.1 which calculates the new revenue if the task was forced to be allocated with a price of zero. The task price is equal to the current server revenue − new revenue with the task allocated + a price change variable to increase the revenue of the server. The minimum price returned by $P(i, k)$ is then compared to the job's maximum reserve price (that would be private in the equivalent decentralised algorithm) to confirm if the job is willing to pay at that price. If the job is willing then the job is allocated to the minimum price server and the job price set to the agreed price. However in the process of allocating a job then the currently allocated jobs on the server could be unallocated so these jobs allocation's and price's are reset then appended to the set of unallocated jobs.

4.4 Attributes of proposed algorithms

In table 3, the important attributes for the proposed algorithm

Attribute	GM	CVA	DIA
Truthfulness		Yes	No
Optimality	No	No	No
Scalability	Yes	Yes	No
Information requirements from users	All	All	Not the reserve value
Communication over heads	Low	Low	High
Decentralisation	No	No	Yes

Table 3: Attributes of the proposed algorithms: Greedy mechanism (GM), Critical Value auction(CVA) and Decentralised Iterative auction (DIA)

5 EMPIRICAL EVALUATION

To test the algorithms presented in section 4, synthetic models have been used to generate a list of tasks and servers.

The synthetic models have been handcrafted with each attribute being generated from a gaussian distribution with a mean and standard deviation.

To compare the greedy algorithm to the optimal elastic allocation, a branch and bound was implemented to solve the problem in section 3.2. In order to compare to fixed speed equivalent models, the minimum total resource required to run the job is found and set as the resource speeds for all of the tasks, with the optimal solution for running the job with the fixed speeds is found as well. To implement the greedy mechanism, the value density function was $\frac{v_j}{s_j+w_j+r_j}$, server selection was $argmin_{\forall i \in I} S_i^{'} + W_i^{'} + R_i^{'}$ and the resource allocation was $min s_j^{'} + w_j^{'} + r_j^{'}$ for job j and servers I.

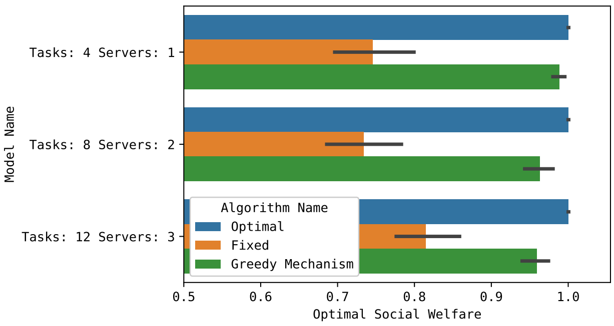


Figure 5: Comparison of the social welfare for the greedy mechanism, optimal, relaxed problem, time limited branch and bound

As figure 5 shows, the greedy mechanism achieves 98% of the optimal solution for the small models, the mechanism achieves within 95% for larger models. In comparison, the fixed allocation achieves 80% of the optimal solution and always does worse than the social welfare of the greedy mechanism.

Figure 6 compares the social welfare of the auction mechanisms: vcg, fixed resource speed vcg, critical value auction and the decentralised iterative auction with different price change variables.

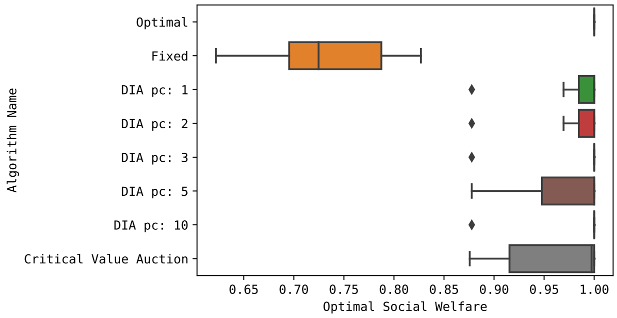


Figure 6: Comparison of the social welfare for the auction mechanisms

VCG is an economically efficient auction that requires the optimal solution to the problem in section 3.2.

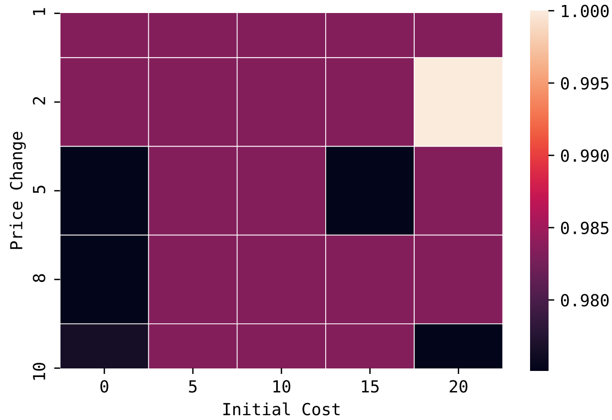


Figure 7: Average number of rounds with a price change variables and task initial cost

Within the context of edge cloud computing, the number of rounds for the decentralised iterative auction is important to making it a feasible auction as it is proportional to the time required to run. We investigated the effect of two heuristic on the number of rounds and social welfare of the auction; the price change variable and initial cost heuristic. With an auction using as minimum heuristic values for the price change and initial cost, figure 7, on average 400 rounds were required for the price to converge while an auction using a price change of 10 and initial cost of 20 means that only on average 80 rounds are required, 5x less. But by using high initial cost and price change heuristics, this can prevent tasks from being allocated, figure 8, shows that the difference in social welfare is only 2% from minimum to maximum heuristics.

6 CONCLUSIONS

In this paper, we studied a resource allocation problem in edge clouds, where resources are elastic and can be allocated to tasks at varying speeds to satisfy heterogeneous requirements and deadlines.

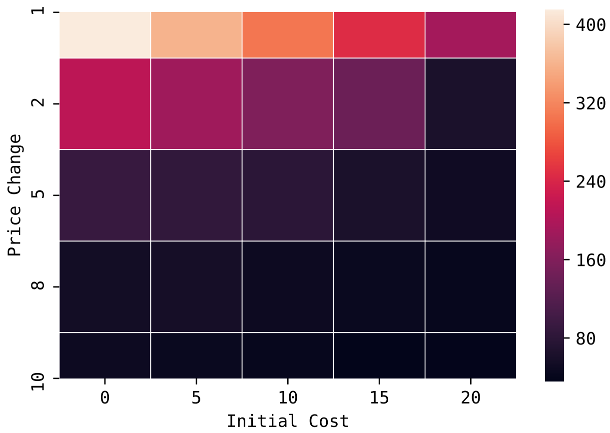


Figure 8: Average social welfare with a price change variables and task initial cost

To solve the problem, we proposed a centralized greedy mechanism with a guaranteed performance bound, and a number of auction-based mechanisms that also consider the elasticity of resources and limit the potential for strategic manipulation. We show that explicitly taking advantage of resource elasticity leads to significantly better performance than current approaches that assume fixed resources.

In future work, we plan to consider the dynamic scenario where tasks arrive and depart from the system over time, and to also consider the case where task preemption is allowed.

REFERENCES

[1] Zubaida Alazawi, Omar Alani, Mohmmad B. Abdljabar, Saleh Altowaijri, and Rashid Mehmood. 2014. A Smart Disaster Management System for Future Cities. In *Proceedings of the 2014 ACM International Workshop on Wireless and Mobile Technologies for Smart Cities (WiMobCity '14)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/2633661.2633670

[2] M. Bahrami. 2015. Cloud Computing for Emerging Mobile Cloud Apps. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. 4–5. https://doi.org/10.1109/MobileCloud.2015.40

[3] Fan Bi, Sebastian Stein, Enrico Gerding, Nick Jennings, and Thomas La Porta. 2019. A truthful online mechanism for resource allocation in fog computing. In *PRICAI 2019: Trends in Artificial Intelligence. PRICAI 2019*, A. Nayak and A. Sharma (Eds.), Vol. 11672. Springer, Cham, 363–376. https://eprints.soton.ac.uk/431819/

[4] Zhiyi Huang Bingqian Du, Chuan Wu. 2019. Learning Resource Allocation and Pricing for Cloud Profit Maximization. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*. 7570–7577.

[5] Edward H. Clarke. 1971. Multipart pricing of public goods. *Public Choice* 11, 1 (01 Sep 1971), 17–33. https://doi.org/10.1007/BF01726210

[6] P. Corcoran and S. K. Datta. 2016. Mobile-Edge Computing and the Internet of Things for Consumers: Extending cloud computing and services to the edge of the network. *IEEE Consumer Electronics Magazine* 5, 4 (2016).

[7] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, and K. S. Chan. 2019. Service Placement and Request Scheduling for Data-intensive Applications in Edge Clouds. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 1279–1287. https://doi.org/10.1109/INFOCOM.2019.8737468

[8] Theodore Groves. 1973. Incentives in Teams. *Econometrica* 41, 4 (1973), 617–631. http://www.jstor.org/stable/1914085

[9] L. Guerdan, O. Apperson, and P. Calyam. 2017. Augmented Resource Allocation Framework for Disaster Response Coordination in Mobile Cloud Environments. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*.

[10] Hans Kellere, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack problems*. Springer.

[11] Dinesh Kumar, Gaurav Baranwal, Zahid Raza, and Deo Prakash Vidyarthi. 2017. A systematic study of double auction mechanisms in cloud computing. *Journal of Systems and Software* 125 (2017), 234 – 255. https://doi.org/10.1016/j.jss.2016.12.009

[12] Y. Liu, F. R. Yu, X. Li, H. Ji, and V. C. M. Leung. 2018. Distributed Resource Allocation and Computation Offloading in Fog and Cloud Networks With Non-Orthogonal Multiple Access. *IEEE Transactions on Vehicular Technology* 67, 12 (2018).

[13] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. 2017. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys Tutorials* 19, 4 (2017).

[14] Kabrane Mustapha, Krit Salah-ddine, and L. Elmaimouni. 2018. Smart Cities: Study and Comparison of Traffic Light Optimization in Modern Urban Areas Using Artificial Intelligence. *International Journal of Advanced Research in Computer Science and Software Engineering* 8 (02 2018), 2277–128. https://doi.org/10.23956/ijarcsse.v8i2.570

[15] Kameng Nip, Zhenbo Wang, and Zizhuo Wang. 2017. Knapsack with variable weights satisfying linear constraints. *Journal of Global Optimization* 69, 3 (01 Nov 2017), 713–725. https://doi.org/10.1007/s10898-017-0540-y

[16] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. 2007. *Algorithmic game theory*. Cambridge university press. 229 pages. https://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf

[17] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. 2007. *Algorithmic game theory*. Cambridge university press. 229–230 pages. https://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf

[18] Mallesh M. Pai and Aaron Roth. 2013. Privacy and Mechanism Design. *SIGecom Exch.* 12, 1 (June 2013), 8–29. https://doi.org/10.1145/2509013.2509016

[19] M. Sapienza, E. Guardo, M. Cavallo, G. La Torre, G. Leombruno, and O. Tomarchio. 2016. Solving Critical Events through Mobile Edge Computing: An Approach for Smart Cities. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*.

[20] G. Sreenu and M. A. Saleem Durai. 2019. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data* 6, 1 (06 Jun 2019), 48. https://doi.org/10.1186/s40537-019-0212-5

[21] William Vickrey. 1961. Counterspeculation, Auctions, and Competitive Sealed Tenders. *The Journal of Finance* 16, 1 (1961), 8–37. http://www.jstor.org/stable/2977633

[22] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. M. Lau. 2017. Online Auctions in IaaS Clouds: Welfare and Profit Maximization With Server Costs. *IEEE/ACM Transactions on Networking* 25, 2 (April 2017), 1034–1047. https://doi.org/10.1109/TNET.2016.2619743

Appendix B: SPIE Presentation

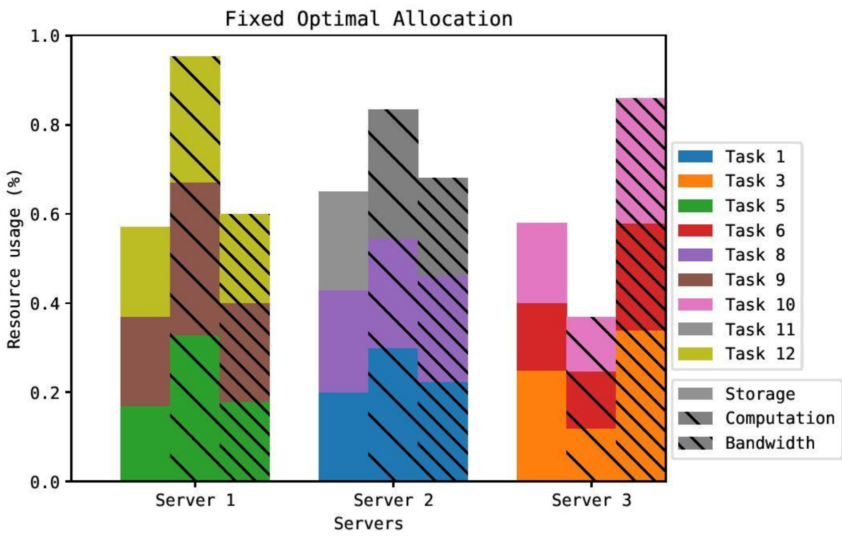
This presentation was produced from the same work as the Towers et al. (2020) with the title ”Analytical agility at the edge of the network through auction mechanisms” at was submitted to the conference on Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II, as part of the SPIE Defense + Commercial Sensing.

Analytical agility at the edge of the network through auction mechanisms

By Mark Towers, Fidan Mehmeti, Sebastian Stein, Tim Norman, Tom La Porta, Caroline Rublein and Geeth De Mel

Motivation

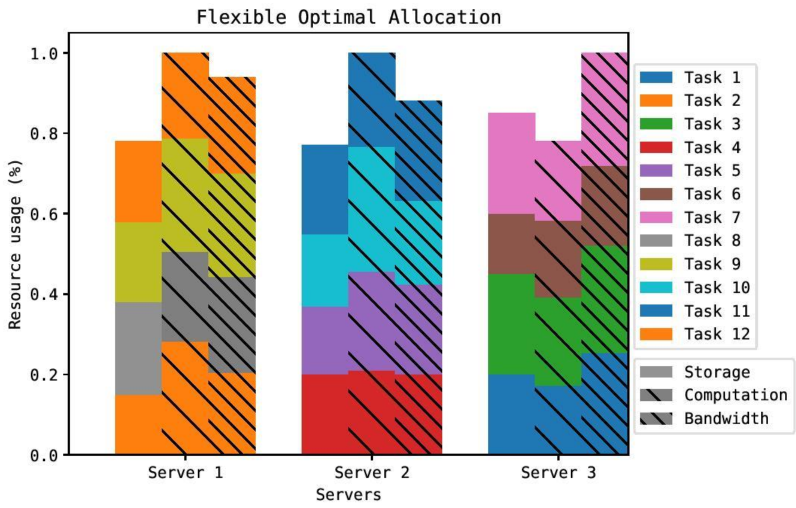
- Edge cloud computing allows coalitions to run computationally demanding analytical tasks in military tactical networks that couldn’t be run locally by the user.
- However, edge cloud computing servers have significantly fewer resources than traditional cloud computing servers.
- They therefore require efficient and effective allocation of these resources to maximise the number of tasks that can be run concurrently
- Previous research considered a fixed allocation scheme where task requested a fixed resource usage
- However, resource bottleneck can easy occur when numerous users over request particular resources, limit the number of tasks that can be run



- Here servers are described with three resources (storage, computation and bandwidth).
- Each task uses a percentage of these resources in order to run being the coloured bars.

Flexible resource allocation mechanism

- Principle – That the time taken for an operation to complete is proportional to the amount of resources allocated
- Instead task submit their resource requirement over their lifetime and a deadline to be computed by
- This is used by servers to determine how resources are allocated to individual tasks
- Algorithm aims is to maximise the social welfare (sum of task values' that are computed within the deadline)
- This proposes research challenges as no previous algorithms exist that are compatible with this mechanism and to deal with self-interested task owners who may wish to misreport their task values and requested resources.



Deadline Constraint

$$\frac{s_j}{s_j'} + \frac{w_j}{w_j'} + \frac{r_j}{r_j'} \leq d_j$$

s_j	Required Storage	s_j'	Loading task speed
w_j	Required computation	s_j'	Compute speed
r_j	Required results data	r_j'	Sending results speed
d_j	Deadline		

Optimisation problem

$$
\begin{aligned}
&\max \sum_{\forall j \in J} v_j \left(\sum_{\forall i \in I} x_{i,j} \right) && && (1)\\
&\text{s.t.} \\
&\sum_{\forall j \in J} s_j x_{i,j} \le S_i, && \forall i \in I, && (2)\\
&\sum_{\forall j \in J} w'_j x_{i,j} \le W_i, && \forall i \in I, && (3)\\
&\sum_{\forall j \in J} (r'_j + s'_j) \cdot x_{i,j} \le R_i, && \forall i \in I, && (4)\\
&\frac{s_j}{s'_j} + \frac{w_j}{w'_j} + \frac{r_j}{r'_j} \le d_j, && \forall j \in J, && (5)\\
&0 \le s'_j \le \infty, && \forall j \in J, && (6)\\
&0 \le w'_j \le \infty, && \forall j \in J, && (7)\\
&0 \le r'_j \le \infty, && \forall j \in J, && (8)\\
&\sum_{\forall i \in I} x_{i,j} \le 1, && \forall j \in J, && (9)\\
&x_{i,j} \in \{0, 1\}, && \forall i \in I, \forall j \in J. && (10)
\end{aligned}
$$

- Using this flexibility principle, an optimisation problem can be mathematically described
- The aim is to maximise the sum of task value that can be computed within the task deadline (equation 1)
- Constraints 2-4 limit the resource usage to be within server capacity
- Constraint 5 forces the task to be completed within its deadline
- Constraints 6-8 force the resource speeds to be positive
- Constraints 9-10 limit a task be allocated to only a single server
- This problem is NP-Hard due to being a knapsack problem

s_j	Required Storage for task j	s_j'	Loading task speed for task j
w_j	Required computation for task j	s_j'	Compute speed for task j
r_j	Required results data for task j	r_j'	Sending results speed for task j
d_j	Deadline for task j	v_j	Value of task j
S_i	Storage capacity of server i	W_i	Computational capacity of server i
R_i	Bandwidth capacity of server i	$X_{i,j}$	Allocation of task j to server i

Approaches

Properties	Greedy mechanism	Critical value auction	Decentralised Iterative auction
Strategyproof	No	Yes	No
Optimal	No	No	No (however this does occur a majority of the time)
Scalability	High	High	Medium
Information requirements from users	All information	All information	All information except the reserved private value
Communications overhead	Low	Low	High
Decentralisation	No	No	Yes

Greedy mechanism

- The algorithm has a lower-bound of 1/n of the optimal welfare. This is as it unknown after the first task is allocated if any subsequent tasks can be allocated. However in practice, this case almost never occurs.
- Algorithm steps:
  1. The list of tasks are sorted in descending order by a value density function.
  2. For each task in the sorted list, a server is selected from the list of available servers using the server selection function.
  3. Then the resource allocated is determined by a resource allocation function for the task on the server
- As a results, the algorithm has polynomial time complexity
- The algorithm however assumes that tasks are not lying about their attributes like value or required resources. This problem is addressed by the critical value auction.

Algorithm 1 Greedy Mechanism

Require: J is the set of tasks and I is the set of servers
Require: $S_i^{'}, W_i^{'}$ and $R_i^{'}$ is the available resources (storage, compu tation and bandwidth respectively) for server i.
Require: $\alpha(j)$ is the value density function of a task
Require: $\beta(j, I)$ is the server selection function of a task and se of servers returning the best server, or $\emptyset$ if the task is not able t be run on any server
Require: $\gamma(j, i)$ is the resource allocation function of a task an server returning the loading, compute and sending speeds
Require: $sort(X, f)$ is a function that returns a sorted list of el ements in descending order, based on a set of elements and function for comparing elements

$J^{'} \leftarrow sort(J, \alpha)$
for all $j \in J^{'}$ **do**
  $i \leftarrow \beta(j, I)$
  if $i \neq \emptyset$ **then**
    $s_j^{'}, w_j^{'}, r_j^{'} \leftarrow \gamma(j, i)$
    $x_{i,j} \leftarrow 1$
  end if
end for

Critical value auction

- Single parameter domain auctions are a well-researched area in mechanism design, with the critical value being the minimum value a buyer must report such that the item is still sold to them.
- Using the critical value, strategyproof (weakly-dominant incentive compatible) auctions can be created using the greedy mechanism previously explained as the method of calculating each task's critical value.
- Auction steps:
  1. The greedy mechanism is run with the reported task values to find the task that would be allocated
  2. For each task that would be allocated, we find the critical value for the task, the user then pays this value instead of the reported value as in the greedy mechanism. The critical value is found by:
     a. Removing the task from the list of sorted tasks (greedy mechanism step 1)
     b. Running the greedy mechanism normally except after each task is allocated checking if the task can still be allocated on any server
     c. Once the task is unable to be allocated to any server, the critical value density is equal to the value density of last task allocated
     d. The critical value is equal to the inverse of the value density function using the critical value density
- Due to the use of the greedy mechanism, the auction inherits the same social welfare performance as the greedy mechanism
- The auction's time complexity is still polynomial as well, as it just computes the greedy mechanism multiple times, up to number of tasks + 1.
- While any value density function can be used, in order for the auction to be strategyproof, the value density function used must be monotonic meaning that if the user misreports a task attribute, the value density must decrease.

$$\frac{v_j d_j}{s_j + w_j + r_j}$$

Decentralised iterative auction

- While the critical value auction is incentive compatible, it requires the revelation of a task's value.
- However, for some users, e.g., in tactical networks, they may not wish to reveal this information to other coalition partners.
- The VCG mechanism works by calculating the price of an auction item by finding the difference in social welfare when the task exists and doesn't exist
- We modify this mechanism to be decentralised instead of centralised such that each server calculates the difference. To do this requires a modified optimisation problem for the server to maximise task prices instead of task values.
- Auction steps:
  1. Unallocated tasks is equal to a initial list of tasks
  2. While unallocated tasks has tasks
     A. Select a random task from the list of unallocated tasks
     B. The task price is the minimum price from all of the servers which calculates its price using the modified optimisation problem
     C. If the task price is greater than the minimum price then the task is disregard otherwise allocate the task to the server
     D. However by allocated the task to the server, other tasks may be kicked off. These task are added back to the unallocated tasks list

Algorithm 2 Decentralised Iterative Auction

Require: I is the set of servers
Require: J is the set of unallocated tasks, which initial is the set of all tasks to be allocated
Require: $P(i, k)$ is solution to the problem in section 4.3.1 using the server i and new task k. The server's current tasks is known to itself and its current revenue from tasks so not passed as arguments.
Require: $R(i, k)$ is a function returning the list of tasks not able to run if task k is allocated to server i
Require: $\leftarrow_R$ will randomly select an element from a set
  while $|J| > 0$ **do**
    $j \leftarrow_R J$
    $p, i \leftarrow argmin_{i \in I} P(i, j)$
    if $p \leq v_j$ **then**
      $p_j \leftarrow p$
      $x_{i,j} \leftarrow 1$
      for all $j' \in R(i, j)$ **do**
        $x_{i,j'} \leftarrow 0$
        $p_{j'} \leftarrow 0$
        $J \leftarrow J \cup j'$
      end for
    end if
    $J \leftarrow J \setminus \{j\}$
  end while

Modified server optimisation problem

- A task's price is equal to the difference in the server revenue (the task doesn't exist) to when the task is included with a price of zero (plus a small value)
- This modifications to the general optimisation problem is that it is only for single server with the new task referred to as n'.
- The resulting constraints are extremely similar except that the new task is forced to be allocated to the server.
    - Objective function is to maximise the sum of computed task prices (Equation 11)
    - Constraints 12-14 limit the resource usage to be within server capacity
    - Constraint 15 forces the task to be completed within its deadline including the new task.
    - Constraints 16-18 force the resource allocation to be positive for all tasks
    - Constraints 19 limits task allocation to be binary

$$\max \sum_{\forall n \in N} p_n x_n \tag{11}$$

s.t.

$$\sum_{\forall n \in N} s_n x_n + s_{n'} \leq S_m, \tag{12}$$

$$\sum_{\forall n \in N} w'_n x_n + w_{n'} \leq W_m, \tag{13}$$

$$\sum_{\forall n \in N} (r'_n + s'_n) \cdot x_n + (r'_{n'} + s'_{n'}) \leq R_m, \tag{14}$$

$$\frac{s_n}{s'_n} + \frac{w_n}{w'_n} + \frac{r_n}{r'_n} \leq d_n, \qquad \forall n \in N \cup \{n'\} \tag{15}$$

$$0 < s'_n < \infty, \qquad \forall n \in N \cup \{n'\} \tag{16}$$

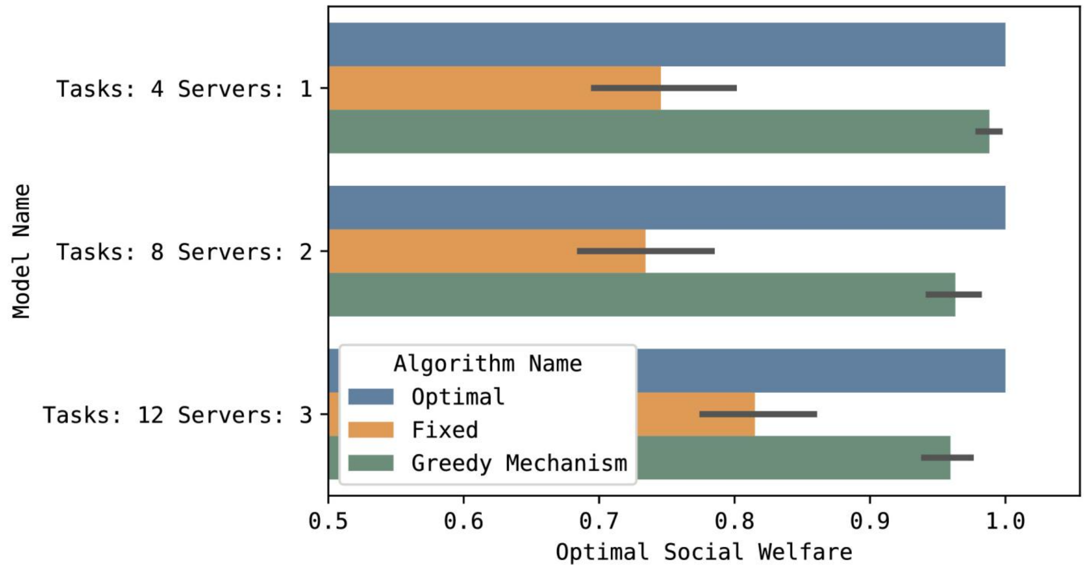
$$0 < w'_n < \infty, \qquad \forall n \in N \cup \{n'\} \tag{17}$$

$$0 < r'_n < \infty, \qquad \forall n \in N \cup \{n'\} \tag{18}$$

$$x_n \in \{0, 1\} \qquad \forall n \in N \tag{19}$$

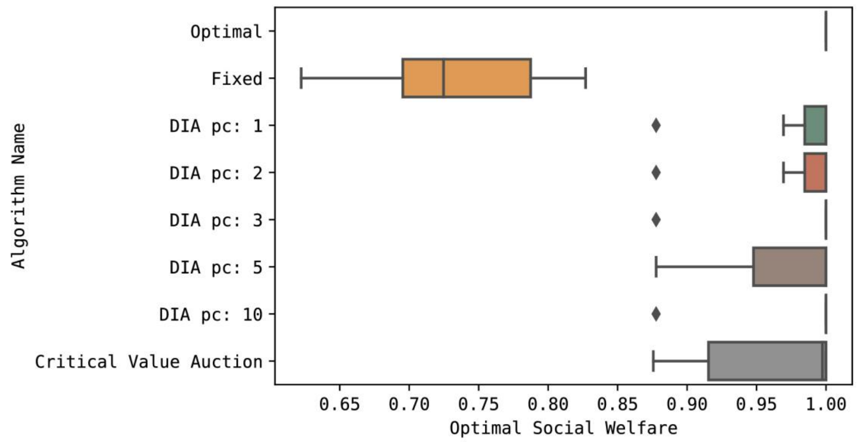
Greedy mechanism and critical value auction results

- To compare our algorithm, we implemented a time-limited optimal solver and a fixed resource allocation mechanism where the task resource usage is fixed between the server preventing any flexibility.
- These results compare the proportion of the optimal social welfare (sum of the computed task values).
- We compared results over a range of environments with different levels of supply and demand.
- With environment that have extremely high demand over a single resources, our system is extremely advantageous as it can deal with this type of pressure that normally is not possible. In cases where server resources are well distributed then this mechanism achieves similar results as the fixed version.



Decentralised Iterative auction

- The VCG auction is an economically efficient auction that finds the optimal allocation to calculate prices. We therefore use the VCG for the optimal flexible and fixed results.
- We found that a majority of the time, the DIA achieves the optimal social welfare however sometimes falls into a local optima
- We also found that the value of the price change doesn't effect the social welfare of the solution much.



Conclusion and future work

- In this work, we have presented a novel resource allocation optimisation problem along with a greedy mechanism to maximise social welfare. Along with two auction mechanisms: critical value auction for strategyproof auctions and a novel decentralised iterative auctions for users who don’t wish to reveal their private task value.
- Future work is to consider an online case of this work as tasks arrive over time instead resources being allocated in batches.

Appendix C: Project management

This is the progress report submitted on the 10th of December 2019.

UNIVERSITY OF SOUTHAMPTON

Faculty of Physical Engineering and Science
School of Electronics and Computer Science

A project report submitted for the award of
MEng Electronic Engineering

Supervisor: Dr Tim Norman

Auctions for online elastic resource allocation in cloud computing

by **Mark Towers**

March 21, 2020

University of Southampton

Abstract

Faculty of Physical Engineering and Science
School of Electronics and Computer Science

A project report submitted for the award of MEng Electronic Engineering

Auctions for online elastic resource allocation in cloud computing

by Mark Towers

Edge clouds enable computational tasks to be completed at the edge of the network, without relying on access to remote data centres. A key challenge in these settings is that limited computational resources often need to be allocated to many self-interested users. Here, existing resource allocation approaches usually assume that tasks have inelastic resource requirements (i.e., a fixed amount of compute time, bandwidth and storage), which may result in inefficient resource use. In this paper, we expand previous work to an online setting such that job will arrive over time with the task prices and resource allocation determined through training an agent using reinforcement learning.

Contents

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a degree at this University;
2. Where any part of this thesis has previously been submitted for any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as: S.R. Gunn. Pdflatex instructions, 2001. URL `http://www.ecs.soton.ac.uk/~srg/softwaretools/document/`
   C. J. Lovell. Updated templates, 2011
   S.R. Gunn and C. J. Lovell. Updated templates reference 2, 2011

Signed:.. Date:..................

Acknowledgements

This project wouldn't have started without Dr Sebastian Stein and a team of Pennsylvania State University that has produced a paper investigating the static case of this problem. So I am grateful for the support they gave in kick starting this project.

My housemates for surviving with me pestering them about proof reading my paper and this project, all of the time.

Chapter 1

Introduction

Google Cloud Platform, Amazon Web Service and Microsoft Azure provide a service to users with computer programs that are too large, difficult or time consuming to be run on standard computer. User can request a fixed amount of resources to run the program, e.g. cpu cores, RAM, hard drive space, bandwidth, etc. However, this can create bottlenecks on certain resources due to large numbers of resource requests preventing other jobs from running. This problem is particularly relevant in edge cloud computing as servers are small thus making the demand on resources much greater. This project considers the case where the user states the total resource requirements for the program instead of the standard procedure that user request a fixed amount of resources. This allows the cloud provider the ability to balance resource demand as it has complete knowledge of all user's requirements and can flexibly change the amount of resources allocated to each task. This can prevent bottlenecks through proper balances of resources allowing more tasks to run simultaneously and can also lower the price due to there being a lower overall demand on resources.

Recently, cloud computing (Bahrami , 2015) has become a popular solution for remotely running data-intensive applications. But for some problem domains, it is not possible to use large cloud providers, for example running highly delay-sensitive tasks or where connectivity to the cloud is intermittent. Mobile edge computing (Mao et al., 2017) has emerged as a complementary paradigm to allow for small data-centers, close to users, to execute tasks. These data centers are known as edge clouds.

Disaster response, smart cities and Internet-of-things (IoT) are popular technologies that utilise mobile edge computing due to the use of ability to process small programs locally with low latency. For smart cities, this allows for the possibly of smart intersections with the use of road-side sensors or smart traffic lights based on cameras to minimise the waiting times (Mustapha et al., 2018). Or for the police to analysis CCTV footage to spot suspicious behaviour or to track people between cameras (Sreenu and Saleem Durai, 2019). In the case of disaster response, maps can be produced using autonomous vehicles sensors to be used in the search for potential victims and support responders (Alazawi et al., 2014).

To compute these task, several types of resources are required included communication bandwidth, computational power and data storage resources (Farhadi et al., 2019). Tasks will have a deadline such that the program must be completed before this point and a private value. This value is depend on the program itself and its value to the owner, .e.g analysis air pollution is less important than preventing traffic jams at rush hour or tracking a criminal on the run. This project is interested in allocated task to servers to maximise the social welfare (sum of all allocated task values) over time. But due to users being self-interested, they may behave strategically (Bi et al., 2019) or prefer to not reveal their value publicity (Pai and Roth, 2013).

The shortcoming of existing work for resource allocation in edge cloud computing (Farhadi et al., 2019; Bi et al., 2019) has the assumption that tasks have fixed resource requirements. However, flexibility is possible in practise with how resources are allocaed to each task. For example, the allocated bandwidth for loading the program is proportional to the time taken to load the program. This is true of also the computational requirements and for sending results back to the user. This project investigates flexible allocation of resource and pricing mechanisms when task arrive over time and have private values.

Chapter 2

Related Works

Due to the novel approach for resource allocation in cloud computing, there is few papers that allow for flexible resource allocation. However there is a considerable amount of research in the area of resource allocation and pricing in cloud computing, some of which use auction mechanisms to deal with competition Kumar et al. (2017); Zhang et al. (2017); Bingqian Du (2019); Bi et al. (2019). In Section 2.1 considers the previously related work for flexible resource allocation in cloud computing and Section 2.2 consider recent work in the field of reinforcement learning.

2.1 Related work in Cloud computing

A majority of the approaches for pricing and resource allocation in cloud computing require users to request a fixed amount of certain resource with the cloud provider having no control over the resources only the servers that the task was allocated to (Kumar et al., 2017; Zhang et al., 2017; Bingqian Du, 2019; Bi et al., 2019). The flexible approach that this project assumed has only been considered in Towers et al. that allows the server to distribute its resources more efficiently based on each task's requirements. The primary difference between this project and that paper is that this project considers the addition of time allowing for resource speed to change over time and that there are task stages.

Previous work by Towers et al. considers three solutions to a single-shot problem case, a greedy algorithm to quickly approximate a solution to maximise the social welfare and two auction mechanisms as server are normally paid for usage of their resources. The greedy algorithm is a polynomial time algorithm that will find solution within $\frac{1}{n}$ of the optimal social welfare. This is done through the use of modular heuristics for ordering the task by density then for each task, select a server based on available resource on each servers then to allocate resources that minimises a resource heuristics. Using certain heuristics, the greedy algorithm achieves at least 90% of the optimal solution and 20% more than optimal solution for fixed resource equivalent problems. A new distributed iterative auction was developed that use a reverse vcg principle to calculate a task price that meant that a task didnt need to reveal its private value also that the auction could be run in a decentralised way. This means that the auction is budget balanced however it is not economically efficient or incentive compatible. The third algorithm is an implementation of a single parameter auctions (Nisan et al., 2007) using the greedy algorithm to find the critical value of a task. Using this mechanism with a monotonic value density heuristic means that the auction is incentive compatible.

Other closely related work on resource allocation in edge clouds Farhadi et al. (2019) considers both the placement of code/data needed to run a specific task, as well as the scheduling of tasks to different edge clouds. The goal there is to maximize the expected rate of successfully accomplished tasks over time. Our work is different both in the setup and the objective function. Our objective is to maximize the value over all tasks. In terms of the setup, they assume that data/code can be shared and they do not consider the elasticity of resources.

2.2 Related work in Reinforcement learning

Supervised learning allows for the training of agents to converge towards a truth value while unsupervised learning allows for the training of agents find pattern for data where no-truth value exist. Reinforcement learning works in the middle ground where truth value exist but are unknown so agent will interact with an environment that depending on certain actions will result in being rewarded. Using this resulted in the first successful ”machine-learning” agent in 1959 with TD-Checking (Samuel, 1988) where the truth value was the difference in two ”neighbouring” checkers boards. Temporal difference, Q-learning, SARSA and other were early training methods for agents using reinforcement learning.

The work of Mnih et al. (2013) developed the usage of these methods much further by coupling them with deep neural network allowing an agent to be trained using the same algorithm to achieve state-of-the-art in 6 of 7 games tried and superhuman scores in 3. This recent work has reinvigorated to the area primarily due to the availability of data to be used and the computational power available. This has allowed Silver et al. (2017) to achieve mastery of the game of Go learning from no human expert to beat the world champion 4 games to 1. With following work expanding to other games like DOTA 2 (OpenAI) beating the world champions and Starcraft 2 (Vinyals et al., 2017) becoming in the top 2% world wide.

Chapter 3

Proposed solution

The problem case presented in Chapter 1 has two stages: auction and resource allocation. These stages are discussed in sections 3.2 and 3.3 respectively.

3.1 Optimisation problem

A sketch of the system is shown in Fig. 3.1. We assume that in the system there is a set of $I = \{1, 2, \ldots, |I|\}$ servers are heterogeneous in all characters. Each server has a fixed availability of resources: storage for the code/data needed to run a task (e.g., measured in GB), computation capacity in terms of CPU cycles per time interval (e.g., measured in FLOP/s), and communication bandwidth to receive the data and to send back the results of the task after execution (e.g., measured in Mbit/s). We denote these resources for server i: the storage capacity as S_i, computation capacity as W_i, and the communication capacity as R_i.

There is a set $J = \{1, 2, \ldots, |J|\}$ of different tasks that require service from one of the servers in set $I = \{1, 2, \ldots, |I|\}$. To run any of these tasks on a server requires storing the appropriate code/data on the same server. These could be, for example, a set of images, videos or CNN layers in identification tasks. The storage size of task j is denoted as s_j with the rate at which the program is transferred to the server i at time t being $s'_{i,j,t}$. For a task to be computed successfully, it must fetch and execute instructions on a CPU. We consider the total number of CPU cycles required for the program to be w_j, where the rate at which the CPU cycles are assigned to the task on server i at time t is $w'_{i,j,t}$. Finally, after

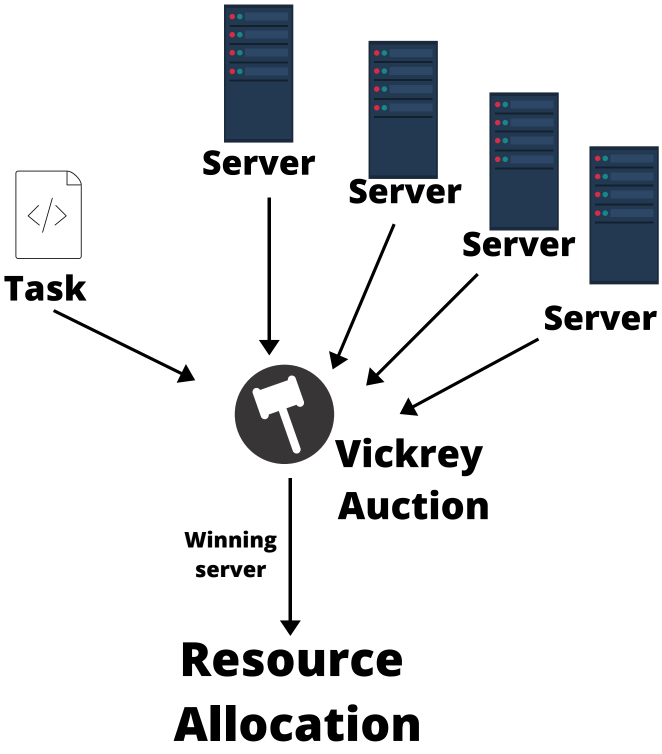


FIGURE 3.1: System model

the task is run and the results obtained, the latter need to be sent back to the user. The size of the results for task j is denoted with r_j, and the rate at which they are sent back to the user is $r^{'}_{i,j,t}$ on server i at time t. Every task has a beginning time, denoted by b_j and a deadline, denoted by d_j. This is the maximum time for the task to be completed in order for the user to derive its value. This time includes: the time required to send the data/code to the server, run it on the server, and get back the results. Therefore for the task to be successfully completed, it must completed fulfill the constraint in equation (3.1). These operations must occur in order (loading, computing then sending of results) as a server couldn't compute a task that was not fully loaded on the machine.

$$\frac{s_j}{\sum_{t=b_j}^{d_j} s^{'}_{i,j,t}} + \frac{w_j}{\sum_{t=b_j}^{d_j} w^{'}_{i,j,t}} + \frac{r_j}{\sum_{t=b_j}^{d_j} r^{'}_{i,j,t}} \leq d_j \qquad \forall j \in J \tag{3.1}$$

As server have limited capacity, the total resource usages for all tasks running on a server must be capped. The storage constraint (equation (3.2)) is unique as the previous amount loaded in kept till the end of a program on server. While the computation capacity (equation (3.3) is the sum of compute used by all of the tasks on a server i at time t and the bandwidth capacity (equation (3.4)) is the sum of loading and sending usages by tasks.

$$\sum_{j \in J} \left(\sum_{t=b_j}^{d_j} s'_{i,j,t} \right) \leq S_i, \qquad \forall i \in I \tag{3.2}$$

$$\sum_{j \in J} w'_{i,j,t} \leq W_i, \qquad \forall i \in I, t \in T \tag{3.3}$$

$$\sum_{j \in J} s'_{i,j,t} + r'_{i,j,t} \leq R_i, \qquad \forall i \in I, t \in T \tag{3.4}$$

$$\tag{3.5}$$

3.2 Auction solution

If an agent wish to run on task on the cloud, the task can be put forward with its requirements of required storage, computation, results data and deadline. In order for fast and truthful, a reverse Vickrey auction (Vickrey, 1961) will be implemented where servers all submit their bid for the task with the winner being the server with the lowest price but actually only gains second lowest price. The Vickrey auction is incentive compatible meaning that the dominant strategy for bidding on a task is to bid your truthful value for a task. This should help server as they dont need to learn how to outbid another agent as it only needs to consider its own evaluation. As there is also only a single round of bidding compared to alternative auctions like English or Dutch auctions, this makes auctioning fast no matter the number of servers and it also allows for a reserve price to be used.

In order to calculate the price of the task for a server requires a understanding the resource requirements of the task, the future supply and demand for tasks and the resource requirements of currently allocated tasks. Due to the complexity in creating a heuristic that can accurately use this information and the amount of memory required for a table based approach. Because of this, a long/short term memory (LSTM) will be implemented (Hochreiter and Schmidhuber, 1997) for evaluating the price of a task. The justified for the use of this network over other neural network models is explained in Section 4.1. The network would take as input, the currently allocated tasks requirements, the possible task requirements and the server resource capacity, outputting just a single value representing the price of the task, normalised between 0 and 100.

3.3 Resource allocation solution

In previous work (Towers et al.), that utilised a single shot problem case where jobs wouldnt arrive over time, the resource speeds set were fixed and assumed that a task loading, computing and sending result occurred concurrently. With the addition of time, results in these assumptions not to hold anymore as tasks contain stages for the loading, computing and sending of results thus requiring allocated resource speeds to change over time. Therefore at each time step, a server needs to reallocate all of its resource to its currently allocated tasks as some tasks will have completely one of its stages.

In order to select how to allocate resource to tasks, this problem doesn't seem as complex as the pricing in section 3.2 therefore simple heuristic and long/short term memory neural network will be implemented and compared. This is justified in section 4.2. The LSTM will take as input, all of the currently allocated tasks that are at a particular stages resource requirements and the task's resource requirement returning a single value between 1 to 100. Once this is completed for each job, the percentage of the total values will be assigned to each task.

3.4 Training and reward schemes

There are three popular types of training methods for neural networks: supervised, unsupervised and reinforcement learning. This project will utilise reinforcement learning as supervised learning requires truth labels for data that for this problem case is too difficult to compute. While Unsupervised learning is generally used for grouping data together in groups making it not appropriate for this project. Therefore reinforcement learning will be utilised as the agent will interact with the environment resulting in actions and can earn rewards through certain actions.

The reward scheme for the pricing heuristic is equivalent to the winning bid however if the task fails to be completed then the negative bid is the reward given at the time the task at auction. This aims to force the heuristic to only bid on tasks that it can complete but not to penalise if the agent fails to win a task in an auction. The agent's future discount variable will stop after the deadline of the task as the reward of the agent winning a task has the largest affect now and it affects shouldn't continue when the task is not allocated.

Resource allocation uses a reward scheme similar to the pricing heuristics except that the reward will be awarded at the point that the task is completed. If the task fails to complete then the reward is negative of the task price and the agent's future discount variable is also similar pricing reward scheme.

Chapter 4

Justification of the approach

The proposed solution in Chapter 3 as two parts explained in section 3.2 and 3.3. This chapter explains the reason for why each section is being solved in its particular way.

As the approaches to pricing and resource allocation heuristics are using neural networks to find the optimal function, table 4.1 has a description of how the networks architectures differ.

Neural Network	Description
Artificial neural networks (ANN) McCulloch and Pitts (1943)	Originally developed as a theoretically approximation for the brain, it was found that for networks with at least one hidden layer that a neural network could approximate any function (Csáji, 2001). This made neural networks extremely helpful for cases where a function would normally to difficult to find the exact function, an ANN could be trained through supervised learning to be a close approximation to the true function.
Recurrent neural network (RNN) Elman (1990)	A major weakness of ANN's is that it must use a fixed input and output making it unusable with text, sound or video where the previous data in important in understanding an input. RNN's extend ANN's to allow for connections to neurons again so that the network is not stateless compared to ANN. This means that individual letters of a words can be passed in with the network "remembering" the previous letter.

Long/Short Term Memory (LSTM) Hochreiter and Schmidhuber (1997)	While RNN's can "remember" previous inputs to the network, it also struggles from the vanishing or exploding gradient problem where gradient tends to zero or infinity making it unuseable. LSTM aims to prevent this by using forget gates that determines how much information the next state will get, allowing for more complexity information to be learnt compared to RNN's
Gated Recurrent unit (GRU) Chung et al. (2014)	GRU are very similar to LSTM, except that they use different wiring and a single less gate, using an update gate instead of a forgot gate. These additional mean that the they run faster and are easier to code than LSTM however are not as expressive allowing for less complex functions to be encoded.
Neural Turing Machine (NTM) Graves et al. (2014)	Inspired by computers, neural turing machines build on LSTM by using an external memory module that instead of memory being inbuild in a neuron. This allows for external observers to understand what is going on much better than LSTM due to its black-box nature.
Differentiable neural computer (DNC) Graves et al. (2016)	An expansion to the NTM where the memory module is scalable in size allowing for additional memory to be added if needed.

TABLE 4.1: Neural network descriptions

4.1 Justification for the auction

The auction stage (discussed in Section 3.2) has two considerations, the auction type and the pricing method.

In auction theory, there are numerous types of auctions that have different properties and uses in different areas. The area in which this project is interested in is single indivisible items as while the item has multiple resource requires, a server is required to buy the task as a single unit. Table 4.2 outlines a description of possible auctions while table 4.3 outline the most important properties that an auction has.

Auction type	Description
English auction	A traditional auction where all participant can bid on a single item with the price slowing ascending till only a single participant is left who pays the final bid price. Due to the number of rounds, this requires a large amount of communication and requires tasks to be auctioned in series.
Dutch auction	The reverse of the English auction where the starting price is higher than anyone is willing to pay with the price slowly dropping till the first participant "jumps in". This can result in sub-optimal pricing if the starting price is not highest enough and the latency can have a large effect on the winner.
Japanese auction	Similar to the English auction except that the auction occurs over a set period of time with the last highest bid being the winner. This means that it has the same disadvantages as the English auction except that there is no guarantee that the price will converge to the maximum. Plus additional factors like latency can have a large effect on the winner that will have a larger affect in the application of this project, edge cloud computing. But this time limit results in the auction taking a fixed amount of time unlike the English or Dutch auctions.
Blind auction	Also known as a First-price sealed-bid auction, all participants submit a single secret bid for an item with the highest bid winning and pays their bid value. As a result there is no dominant strategy (not incentive compatible) as an agent would not wish to bid higher than their task evaluation but if all other agents bid significantly lower then it would have been beneficial for the agent to bid much lower than their true evaluation. Due there being a single round of biding, latency doesn't affect an agent and many more auctions could occur within the same time a English, Dutch or Japanese auction would take to run.

Vickrey auction (Vickrey, 1961)	Also known as a second-price sealed bid auction, all participants submit a single secret bid for an item with the highest bid winning but it only pays the price of the second highest bid. Because of this, it is a dominant strategy for an agent to bid its true value as even if the bid is much higher than all other participants its doesn't matter.

TABLE 4.2: Descriptions of auctions

Auction	Incentive compatible	Iterative	Fixed time length
English	False	True	False
Japanese	False	True	True
Dutch	False	True	False
Blind	False	False	True
Vickrey	True	False	True

TABLE 4.3: Properties of the auctions described in Table 4.2

Due to the properties of the Vickrey auction (table 4.3), I believe that it is the best auction to be used. The greatest advantage of the auction is that it is strategyproof meaning the dominant strategy is to truthful bid its price. This means that agents don't have to learn a strategy as with the blind auction where the agent must learn to bid only just lower than other agents. Another advantage of the auction is that it is not iterative, making the auction fast with only a single round and can give a fixed time limit from the task being published to all server bids to be submitted.

However, the standard Vickrey auction will not be used as the task is buying the resources from a server not a server buying the task. But due to resource allocation, the server must bid on the task so the Vickrey auction implemented will work in reverse so the lowest bid will win and the task must pay the second-lowest bid. In the final report, a proof will be provided to show that a reverse Vickrey auction is still incentive compatible.

The second part of the auction solution is the pricing heuristic. I believe that the pricing heuristic would be too complex to encoded into an algorithm if by hand due its need to understand: future resource allocation of currently allocated jobs and the resource requirements of the task. Therefore due to neural network being able to approximate any function (Csáji, 2001) and reinforcement learning methods to training without truth data (Section 2.2). I have outlined in Table 4.1 the properties of popular neural network architectures that would allow for a variable amount of inputs (except for ANN). This is due to having to input to the network the currently allocated tasks to a server that till compute time is of unknown length. Of the available architecture, I predict the Long/Short term memory model is the simplest model that will require the least training but still with the complexity to encode the heuristic. With the Neural Turing Machine and Differentiable Neural Network, these networks are extremely complex and require a large amount of data to train the networks. Also the ability of these networks to be able to store data in external storage is not important as the data doesn't need to be store for future inputs. The opposite problem exists for the Recurrent neural network or the Gated Recurrent unit that they are possibly not complex enough for the pricing heuristic.

4.2 Justification for resource allocation

The justification for the resource allocation neural network choice is very similar justification to the previous section (section 4.1). Long/short term memory architecture should be complex enough for the resource allocation but it is possible that the abilily to use external storage of Neural Turing machine and Differentiable Neural network to store the allocation of resource to previous tasks. But I don't believe that this additional complexity will allow for the heuristic to do later better but it could be investigated in future work.

The reason that the output of the neural network is normalised is done as it would require the network to learn less compared to if the network has output the amount of the available resources for a task. Whereas in a normalised value, the network can output how "important" allocation of resources are for a task not the exact amount of resources allocated.

Chapter 5

Work requirements

For the project, the additional support I will require is more compute power for training of the neural networks. Because of this, I will request access to Iridis 4 with GPUs.

5.1 Work to date

As this project is an extension to previous work done in the Agent, Interaction and Complexity research labs that has produced the paper in Section 5.2. The majority of this research occurred over the summer of 2019 with the paper that is currently under peer review done from October 2019 to 15th November 2019. The paper produced was done with support from Dr Fidan Mehmeti and Dr Sebastian Stein with myself being the primary author.

For the remaining time, I have studied reinforcement learning that is the primary technological additional that will be used in the proposed solution (section 3) and described in Section 2.2.

5.2 Plan of the remaining work

Due to this term having been completing the paper (Section 5.2), I have not done any programming towards the project. Therefore the begin of the next term will be spend building the framework for which different pricing and resource allocation heuristics can be applied and compared. Once this has been

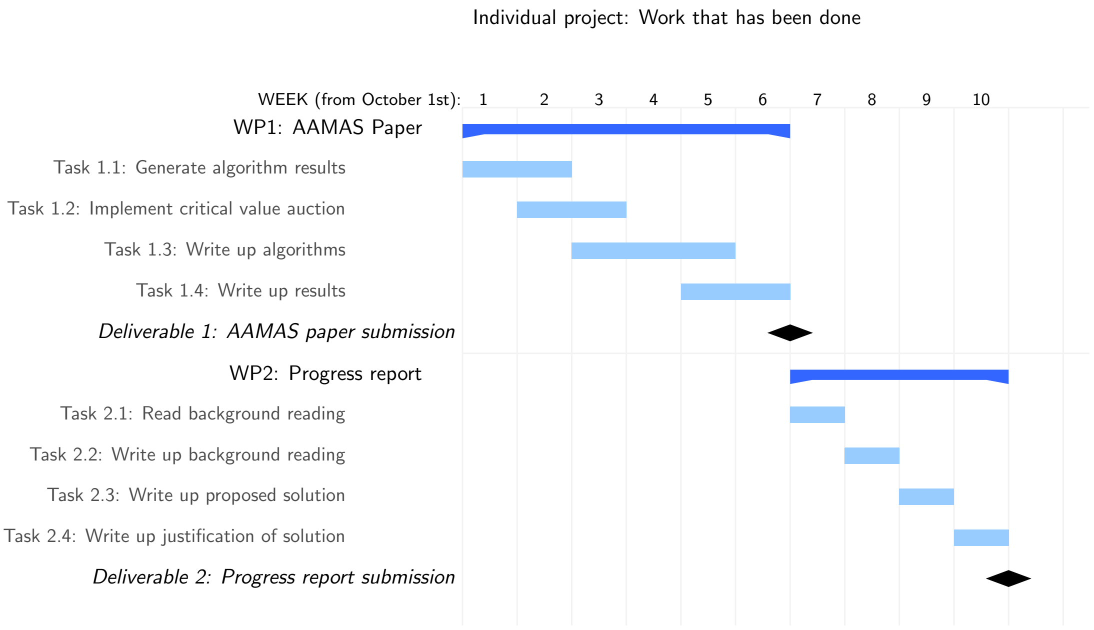


FIGURE 5.1: Work that has been done to date

completed, analysis and comparison of the heuristic will be done with different server and task models. Resulting in a final paper.

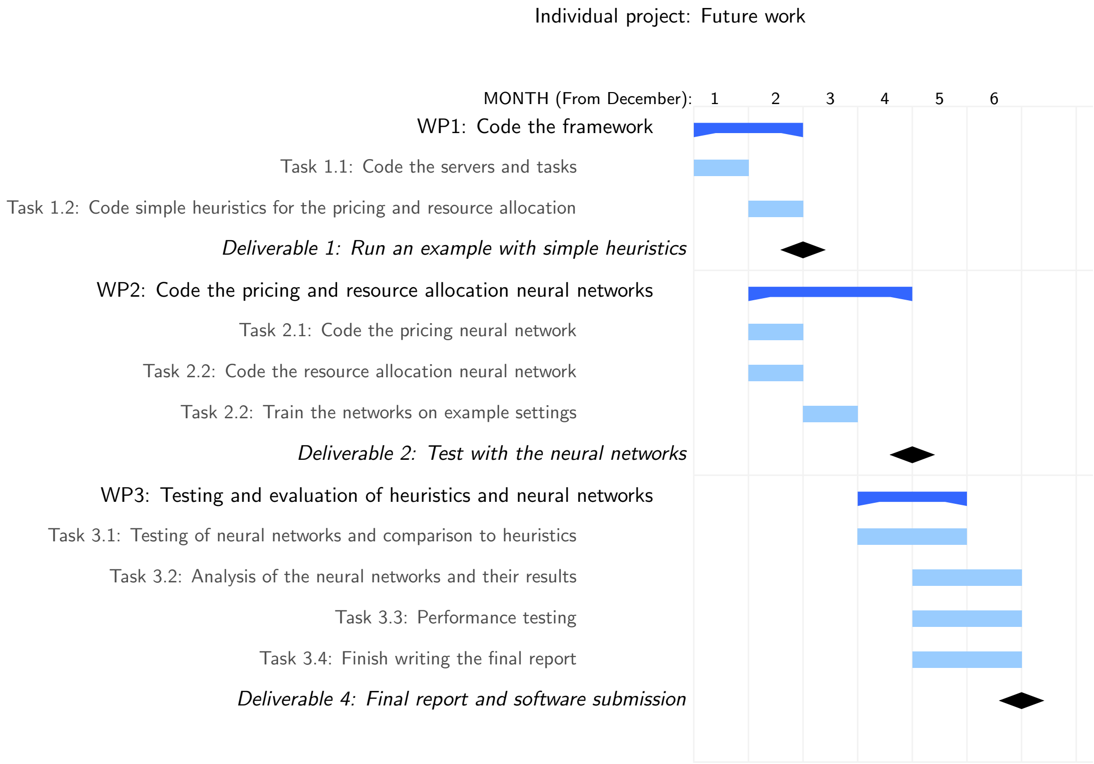


FIGURE 5.2: Work that will be done in the future

Appendices

Appendix A: Paper

This paper has been submitted to the International Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2020 at the University of Auckland. The paper is under peer-review with the authors being myself, Sebastian Stein, Tim Norman, Fidan Mehmeti, Tom La Porta, Caroline Rubein and Geeth Demel and within this project is referred to as Towers et al.. A copy of the paper is found below.

Auction-based Mechanisms for Allocating Elastic Resources in Edge Clouds

Paper #1263

ABSTRACT

Edge clouds enable computational tasks to be completed at the edge of the network, without relying on access to remote data centres. A key challenge in these settings is that limited computational resources often need to be allocated to many self-interested users. Here, existing resource allocation approaches usually assume that tasks have inelastic resource requirements (i.e., a fixed amount of compute time, bandwidth and storage), which may result in inefficient resource use. To address this, we propose a novel approach that takes advantage of the elastic nature of some of the resources, e.g., to trade off computation speed with bandwidth if this allows a server to execute more tasks by their deadlines. We describe this problem formally, show that it is NP-hard and then propose a scalable approximation algorithm. To deal with the self-interested nature of users, we show how to design a centralized auction that incentives truthful reporting of task requirements and values. Moreover, we propose novel auction-based decentralized approaches that are not always truthful, but that limit the information required from users and that can be adjusted to trade off convergence speed with solution quality. In extensive simulations, we show that considering the elasticity of resources leads to a gain in utility of around 20% compared to existing fixed approaches and that our novel auction-based approaches typically achieve 95% of the theoretical optimal.

KEYWORDS

Edge clouds; elastic resources; auctions

1 INTRODUCTION

In the last few years, cloud computing [2] has become a popular solution to run data-intensive applications remotely. However, in some application domains, it is not feasible to rely a remote cloud, for example when running highly delay-sensitive and computationally-intensive tasks, or when connectivity to the cloud is intermittent. To deal with such domains, *mobile edge computing* [13] has emerged as a complementary paradigm, where computational tasks are executed at the edge of mobile networks at small data-centers, known as *edge clouds*.

Mobile edge computing is a key enabling technology for the Internet-of-Things (IoT) [6] and in particular applications in smart cities [19] and disaster response scenarios [9]. In these applications, low-powered devices generate computational tasks and data that have to be processed quickly on local edge cloud servers. More specifically, in smart cities, these devices could be smart intersections that collect data from road-side sensors and vehicles to produce an efficient traffic light sequence to minimize waiting times [14]; or it could be CCTV cameras that analyse video feeds for suspicious behaviour, e.g., to detect a stabbing or other crime in progress [20]. In disaster response, sensor data from autonomous vehicles (including video, sonar and LIDAR) can be aggregated in real time to produce maps of a devastated area, search for potential victims and help first responders in focusing their efforts to save lives [1].

To accomplish these tasks, there are typically several types of resources that are needed, including communication bandwidth, computational power and data storage resources [7], and tasks are generally delay-sensitive, i.e., have a specific completion deadline. When accomplished, different tasks carry different values for their owners (e.g., the users of IoT devices or other stakeholders such as the police or traffic authority). This value will depend on the importance of the task, e.g., analysing current levels of air pollution may be less important than preventing a large-scale traffic jam at peak times or tracking a terrorist on the run. Given that edge clouds are often highly constrained in their resources [12], we are interested in allocating tasks to edge cloud servers to maximize the overall social welfare achieved (i.e., the sum of all task values). This is particularly challenging, because users in edge clouds are typically self-interested and may behave strategically [3] or may prefer not to reveal private information about their values to a central allocation mechanism [18].

An important shortcoming of existing work looking at resource allocation in edge clouds, e.g., [3, 7], is that it assumes tasks have strict resource requirements — that is, each task consumes a fixed amount of computation (CPU cycles per time), takes up a fixed amount of bandwidth to transfer data and uses up a fixed amount of storage on the server. However, in practice, edge cloud servers have some flexibility in how they allocate limited resources to each task. In more detail, to execute a task, the corresponding data and/or code first has to be transferred to the server it is assigned to, requiring some bandwidth. This then takes up storage on the server. Next, the task needs computing power from the server in terms of CPU cycles per time. Once computation is complete, the results have to be transferred back to the user, requiring further bandwidth. Now, while the the storage capacity at the server for every task is *strict*, since the task cannot be run unless all the data are stored, the bandwidth allocation and the speed at which the task is run on the server are *elastic*. The latter two depend on how tight the task's deadline is, and can be adjusted accordingly, so that more tasks can receive service simultaneously. Allocating the elastic resources optimally is the focus of this paper.

Against this background, we make the following novel contributions to the state of the art:

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 2020, Auckland, New Zealand

https://doi.org/doi

- We formulate an optimization problem for assigning the tasks to the servers, whose objective is to maximize total social welfare, taking into account resource limitations and allowing elastic allocation of resources.
- We prove that the problem is NP-hard and propose an approximation algorithm with a performance guarantee of $\frac{1}{n}$, where n is the number of tasks, and a linearithmic computational complexity, i.e., $O(n \log(n))$.
- We propose a range of auction-based mechanisms to deal with the self-interested nature of users. These offer various trade-offs regarding truthfulness, optimality, scalability, information requirements from users, communication overheads and decentralization.
- Using extensive realistic simulations, we compare the performance of our algorithm against other benchmark algorithms, and show that our algorithm outperforms all of them, while at the same time being within 95% to the optimal solution.

The paper is organized as follows. In the next section we discuss related work. This is followed by the problem formulation in Section 3. Our novel resource allocation mechanisms are presented in Section 4. In Section 5, we evaluate the performance of our mechanisms and compare them against the optimal solution and other benchmarks. Finally, Section 6 concludes the work.

2 RELATED WORK

There is a considerable amount of research in the area of resource allocation and pricing in cloud computing, some of which use auction mechanisms to deal with competition [3, 4, 11, 22]. However, these approaches assume that users request a fixed amount of resources system resources and processing rates, with the cloud provider having no control over the speeds, only the servers that the task was allocated to. In our work, tasks' owners report deadlines and overall data and computation requirements, allowing the edge cloud server to distribute its resources more efficiently based on each task's requirements.

Our problem is related to multidimensional knapsack problems. In particular, Nip et al. [15] consider flexibility in the allocation, with linear constraints that are used for elastic weights. The paper provides a pseudo-polynomial time complexity algorithm for solving this problem to maximize the values in the knapsack. Our problem case is similar to their problem, but our problem has non-linear constraints due to the deadline constraint, so their algorithm cannot be applied here.

Other closely related work on resource allocation in edge clouds [7] considers both the placement of code/data needed to run a specific task, as well as the scheduling of tasks to different edge clouds. The goal there is to maximize the expected rate of successfully accomplished tasks over time. Our work is different both in the setup and the objective function. Our objective is to maximize the value over all tasks. In terms of the setup, they assume that data/code can be shared and they do not consider the elasticity of resources.

3 PROBLEM FORMULATION

In this section we first describe the system model. Then, we present the optimization problem and prove its NP-hardness.

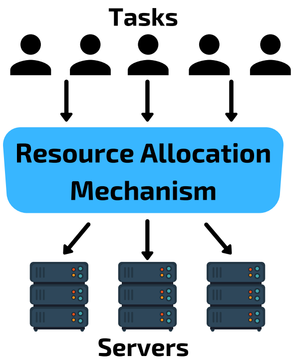


Figure 1: System Model

3.1 System model

A sketch of the system is shown in Fig. 1. We assume that in the system there is a set of servers $I = \{1, 2, \dots, |I|\}$ servers, which could be edge clouds that can be accessed either through cellular base stations or WiFi access points (APs). Servers have different types of resources: storage for the code/data needed to run a task (e.g., measured in GB), computation capacity in terms of CPU cycles per time interval (e.g., measured in FLOP/s), and communication bandwidth to receive the data and to send back the results of the task after execution (e.g., measured in Mbit/s). We assume that the servers are heterogeneous in all their characteristics. More formally, we denote the storage capacity of server i with S_i, computation capacity with W_i, and the communication capacity with R_i.

There is a set $J = \{1, 2, \dots, |J|\}$ of different tasks that require service from one of the servers.[1] Every task $j \in J$ has a value v_j that represents the value of running the task to its owner. To run any of these tasks on a server requires storing the appropriate code/data on the same server. These could be, for example, a set of images, videos or CNN layers in identification tasks. The storage size of task j is denoted as s_j with the rate at which the program is transferred to the server being s_j'. For a task to be computed successfully, it must fetch and execute instructions on a CPU. We consider the total number of CPU cycles required for the program to be w_j, where the rate at which the CPU cycles are assigned to the task per unit of time is w_j'. Finally, after the task is run and the results obtained, the latter need to be sent back to the user. The size of the results for task j is denoted with r_j, and the rate at which they are sent back to the user is r_j'. Every task has its deadline, denoted by d_j. This is the maximum time for the task to be completed in order for the user to derive its value. This time includes: the time required to send the data/code to the server, run it on the server, and get back the results. We assume that there is an *all* or *nothing* task execution reward scheme, meaning that for the task value to be awarded the entire task must be run and the results sent back within the deadline.

[1]We focus on a single-shot setting in this paper. In practice, an allocation mechanism would repeat the allocation decisions described here over regular time intervals, with longer-running tasks re-appearing on consecutive time intervals. We leave a detailed study of this to future work.

3.2 Optimization problem

Given the aforementioned assumptions, the optimal assignment of tasks to servers and optimal allocation of resources in a server to the tasks assigned to that server is obtained as a solution to the following optimization problem. Here, the decision variables are $x_{i,j} \in \{0,1\}$ (whether to run task j on server i) as well as $s_j^{'}$, $r_j^{'}$ and $w_j^{'}$ (indicating the bandwidth rates for transferring the code, for returning the results and the CPU cycles per unit of time, respectively).

$$\max \sum_{\forall j \in J} v_j \left(\sum_{\forall i \in I} x_{i,j} \right) \tag{1}$$

s.t.

$$\sum_{\forall j \in J} s_j x_{i,j} \leq S_i, \qquad \forall i \in I, \tag{2}$$

$$\sum_{\forall j \in J} w_j^{'} x_{i,j} \leq W_i, \qquad \forall i \in I, \tag{3}$$

$$\sum_{\forall j \in J} (r_j^{'} + s_j^{'}) \cdot x_{i,j} \leq R_i, \qquad \forall i \in I, \tag{4}$$

$$\frac{s_j}{s_j^{'}} + \frac{w_j}{w_j^{'}} + \frac{r_j}{r_j^{'}} \leq d_j, \qquad \forall j \in J, \tag{5}$$

$$0 \leq s_j^{'} \leq \infty, \qquad \forall j \in J, \tag{6}$$

$$0 \leq w_j^{'} \leq \infty, \qquad \forall j \in J, \tag{7}$$

$$0 \leq r_j^{'} \leq \infty, \qquad \forall j \in J, \tag{8}$$

$$\sum_{\forall i \in I} x_{i,j} \leq 1, \qquad \forall j \in J, \tag{9}$$

$$x_{i,j} \in \{0,1\}, \qquad \forall i \in I, \forall j \in J. \tag{10}$$

The objective (Eq.(1)) is to maximize the total value over all tasks (i.e., the social welfare). Task j will receive the full value v_j only if it is executed entirely and the results are obtained within the deadline for that task. Constraint (Eq.(2)) relates to the finite storage capacity of every server to store code/data for the tasks that are to be run. The finite computation capacity of every server is expressed through Eq.(3), whereas Eq.(4) denotes the constraint on the communication capacity of the servers. As can be seen, the communication bandwidth comprises two parts: one part to send the data/code or request to the server, and the other part to get the results back to the user.[2] Constraint Eq.(5) is the deadline associated with every task, where the total time of the task in the system is the sum of the time to send the request and code/data to the server, time to run the task, and the time it takes the server to send all the results to the user. Note that if a task is not run on any server, this constraint can be satisfied by choosing arbitrarily high bandwidth and CPU rates (without being constrained by the resource limits of any server). The rates at which the code is sent, run and the results are sent back are all positive and finite (Eqs. (6), (7), (8)). Further, every task is served by at most one server (Eq.(9)). Finally, a task is either served or not (Eq.(10)).

[2]Not that sending and receiving data will not always overlap, but for tractability we assume they deplete a common limited bandwidth resource per time step. This ensures that the bandwidth constraint is always satisfied in practice.

Complexity: In the following we show that this optimization problem is NP-hard.

THEOREM 3.1. *The optimization problem* (1)-(10) *is NP-hard.*

PROOF. The optimization problem without constraint (5) is a 0-1 multidimensional knapsack problem [10], which is a generalization of a simple 0-1 knapsack problem. The latter is an NP-hard problem [10]. Given this, it follows that the 0-1 multidimensional knapsack problem is also NP-hard. Since optimization problem (1)-(10) is a generalization of a 0-1 multidimensional knapsack problem, it follows that it is NP-hard as well. □

Before we propose our novel allocation mechanisms for the allocation problem with elastic resources, we briefly outline an example that illustrates why considering this elasticity is important. In this example, there are 12 potential tasks and 3 servers (the exact settings can be found in table 2 for the tasks and table 1 for the servers).

Figure 2 shows the best possible allocation if tasks have fixed resource requirements. The resource speeds were chosen such to the minimum total resource usage that the task would require from the deadline. Here, 9 of the tasks are run, resulting in a total social welfare of 980 due to the limitation of the server's computation and the task requirement not being balanced.

In contrast to this, Figure 3 depicts the optimal allocation if elastic resources are considered. Here, it is evident that all of the resources are used by the servers whereas the fixed (in figure 2) cant do this. In total, the elastic approach manages to schedule all 12 tasks within the resource constraints, achieving a total social welfare of 1200 (an 19% improvement over the fixed approach).

The figures represent resource usage of the servers by the three bars relating to each of this resources (storage, CPU and bandwidth). For each task that is allocated to the server, the percentage of the resource's used is bar size. Then, for the tasks that are assigned to corresponding servers, the percentage of used resources are also depicted.

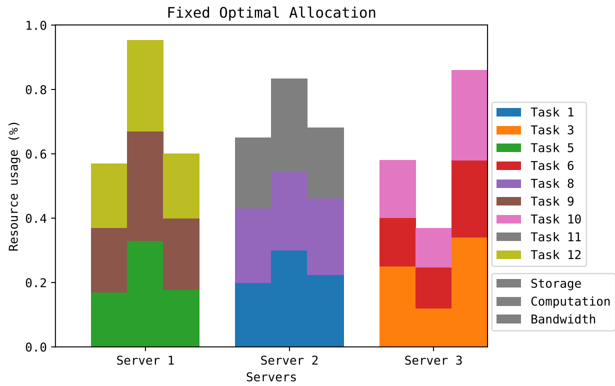


Figure 2: Optimal solution with fixed resources. Due to not being able to balance out the resources, bottlenecks on the server 1 and 2's computation have occurred

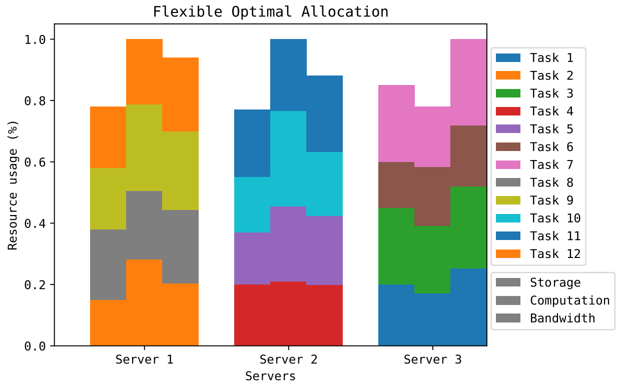


Figure 3: Optimal solution with elastic resources. Compared to the fixed allocation, the elastic allocation is able to fully use all of its resources

Name	S_i	W_i	R_i
Server 1	400	100	220
Server 2	450	100	210
Server 3	375	90	250

Table 1: Servers - Table of server attributes

Name	v_j	s_j	w_j	r_j	d_j	s_j'	w_j'	r_j'
Task 1	100	100	100	50	10	30	27	17
Task 2	90	75	125	40	10	22	32	15
Task 3	110	125	110	45	10	34	30	17
Task 4	75	100	75	35	10	27	21	13
Task 5	125	85	90	55	10	24	28	17
Task 6	100	75	120	40	10	20	32	16
Task 7	80	125	100	50	10	31	30	19
Task 8	110	115	75	55	10	30	22	20
Task 9	120	100	110	60	10	27	29	24
Task 10	90	90	120	40	10	25	30	17
Task 11	100	110	90	45	10	30	26	16
Task 12	100	100	80	55	10	24	24	22

Table 2: Tasks - Table of task attributes, the columns for resource speeds (s_j', w_j', r_j') is for fixed speeds which the flexible allocation does not take into account. The fixed speeds is the minimum required resources to complete the task within the deadline constraint.

4 FLEXIBLE RESOURCE ALLOCATION MECHANISMS

In this section, we propose several mechanisms for solving the resource allocation problem with elastic resources. First, we discuss a centralized greedy algorithm (detailed in Section 4.1) with a $\frac{1}{|J|}$ performance guarantee and polynomial run-time. Then, we consider settings where task users are self-interested and may either report their task values and requirements strategically or may wish to limit the information they reveal to the mechanism. To deal with such cases, we propose two auction-based mechanisms, one of which can be executed in a decentralized manner (in Sections 4.2 and 4.3).

4.1 Greedy Mechanism

As solving the allocation problem with elastic resources is NP-hard, we here propose a greedy algorithm (Algorithm 1) that considers tasks individually, based on an appropriate prioritisation function.

More specifically, the greedy algorithm does this in two stages; the first sorts the tasks and the second allocates them to servers. A value density function is applied to each of the task based on its attributes: value, required resources and deadlines. Stage one uses this function to sort the list of tasks. The second stage then iterates through the tasks in the given order, applying two heuristics to each task: one to select the server and another to allocate resources. The first of these heuristics, called the server selection heuristic, works by checking if a server could run the task if all of its resources were to be used for meeting the deadline constraint (eq 5) then calculating how good it would be for the job to be allocated to the server. The second heuristic, called the resource allocation heuristic, finds the best permutations of resources to minimise a formula, i.e., the total percentage of server resources used by the task.

In this paper we prove that the lower bound of the algorithm is $\frac{1}{|J|}$ (where $|J|$ is the number of jobs) using the value of a task as the value density function and using any feasible server selection and resource allocation heuristic. However we found that the task value heuristic is not the best heuristic as it does not consider the effect of the deadline or resources used for a job. In practice, the following heuristic often works better: $\frac{v_j \cdot (s_j + w_j + r_j)}{d_j}$. For the server selection heuristic we use $argmin_{\forall i \in I} S_i' + W_i' + R_i'$, where S_i', W_i', R_i' are the server's available storage, computation and bandwidth resources respectively. While for the resource allocation heuristic we use $min \frac{W_i'}{w_j'} + \frac{R_i'}{s_j' + r_j'}$.

Theorem 4.1. *The lower bound of the greedy mechanism is $\frac{1}{n}$ of the optimal social welfare*

Proof. Taking the value of a task as the value density function, the first task allocated will have a value of at least $\frac{1}{n}$ total values of all jobs. As the allocation of resources for a task is not optimal, allocation of subsequent tasks is not guaranteed. Therefore, as the optimal social welfare must be the total values of all jobs or lower then the lower bound of the mechanism must be $\frac{1}{n}$ of the optimal social welfare. □

In figure 4, an example allocation using the algorithm is shown using the model from tables 1 and 2. The algorithm uses the recommend heuristic proposed above and allows for all tasks to be allocated achieving 100% of the flexible optimal in figure 3.

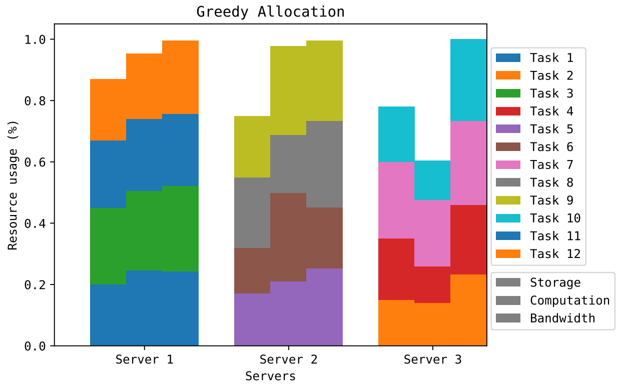


Figure 4: Example Greedy allocation using model from table 2 and 1

Algorithm 1 Greedy Mechanism

Require: J is the set of tasks and I is the set of servers
Require: $S_i^{'}$, $W_i^{'}$ and $R_i^{'}$ is the available resources (storage, computation and bandwidth respectively) for server i.
Require: $\alpha(j)$ is the value density function of a task
Require: $\beta(j, I)$ is the server selection function of a task and set of servers returning the best server, or $\emptyset$ if the task is not able to be run on any server
Require: $\gamma(j, i)$ is the resource allocation function of a task and server returning the loading, compute and sending speeds
Require: $sort(X, f)$ is a function that returns a sorted list of elements in descending order, based on a set of elements and a function for comparing elements

$J^{'} \leftarrow sort(J, \alpha)$
for all $j \in J^{'}$ **do**
  $i \leftarrow \beta(j, I)$
  if $i \neq \emptyset$ **then**
    $s_j^{'}, w_j^{'}, r_j^{'} \leftarrow \gamma(j, i)$
    $x_{i,j} \leftarrow 1$
  end if
end for

Theorem 4.2. *The time complexity of the greedy algorithm is $O(|J|\,|I|)$, where $|J|$ is the number of tasks and $|I|$ is the number of servers. Assuming that the value density and resource allocation heuristics have constant time complexity and the server selection function is $O(|I|)$.*

Proof. The time complexity of the stage 1 of the mechanism is $O(|J| \log(|J|))$ due to sorting the tasks and stage 2 has complexity $O(|J|\,|I|)$ due to looping over all of the tasks and applying the server selection and resource allocation heuristics. Therefore the overall time complexity is $O(|J|\,|I| + |J| \log(|J|) = O(|J|\,|I|)$. □

4.2 Critical Value Auction

Due to the problem case being non-cooperative, if the greedy mechanism was used to allocate resources such that the value is the price paid. This is open to manipulation and misreporting of task attributes like the value, deadline or resource requirements. Therefore in this section we propose an auction that is weakly-dominant for tasks to truthfully report it attributes.

Single-Parameter domain auctions are extensively studied in mechanism design [16] and are used where an agent's valuation function can be represented as single value. The task price is calculated by finding the task's value such that if the value were any smaller, the task could not be allocated. This value is called the critical value. This has been shown to be a strategyproof [17] (weakly-dominant incentive compatible) auction so it is a weakly-dominant strategy for a task to honestly reveal its value.

The auction is implemented using the greedy mechanism from section 4.1 to find an allocation of tasks using the reported value. Then for each task allocated, the last position in the ordered the task list such that the task would still allocated is found. The critical value of the task is then equal to the inverse of the value density function where the density is the density of the next task in the list after that position.

In order that the auction is strategyproof, the value density function is required to be monotonic so that misreporting of any task attributes will result in the value density decreasing. Therefore a value density function of the form $\frac{v_j d_j}{\alpha(s_j, w_j, r_j)}$ must be used so that the auction is strategyproof.

Theorem 4.3. *The value density function $\frac{v_j d_j}{\alpha(s_j, w_j, r_j)}$ is monotonic for task j assuming the function $\alpha(s_j, w_j, r_j)$ is monotonic decreasing.*

Proof. In order to misreport the task private value and deadline must be less than the true value. The opposite is true for the required resources (storage, compute and result data) with the misreported value being greater than the true value. Therefore the α function will increase as the resource requirements increase as well, meaning that density will decrease. □

4.3 Decentralised Iterative Auction

VCG (Vickrey-Clark-Grove) auction [21] [5] [8] is proven to be economically efficient, budget balanced and incentive compatible. A task's price is found by the difference of the social welfare for when the task exists compared to the social welfare when the task doesn't exist. Our auction uses the same principle for pricing by finding the difference between the current server revenue and the revenue when the task is allocated (at £0).

The auction iteratively lets a task advertise its requirements to all of the servers who respond with their price for the task. This price is equal to the server's current revenue minus the solution to the the problem in section 4.3.1 plus a small value called the price change variable. Being the reverse of the VCG mechanism, such that the price is found for when the task exists rather than when it doesn't exist. The price change variable allows for the increase in the revenue of the server and is can be chosen by the server. Once all of the server have responded, the task can compare the minimum server price to its private value. If the price is less then the task will accept the servers with the minimum price offer, otherwise the task will stop looking as the price for the task to run on any server is greater than its reserve price.

To find the optimal revenue for a server m given a new task p and set of currently allocated tasks N has a similar formulation to section 3.2. With an additional variable is considered, a task's price being p_n for task n.

4.3.1 Server problem case.

$$\max \sum_{\forall n \in N} p_n x_n \tag{11}$$

$$\text{s.t.} \tag{12}$$

$$\sum_{\forall n \in N} s_n x_n + s_p \leq S_m, \tag{13}$$

$$\sum_{\forall n \in N} w_n^{'} x_n + w_p \leq W_m, \tag{14}$$

$$\sum_{\forall n \in N} (r_n^{'} + s_n^{'}) \cdot x_n + (r_p^{'} + s_p^{'}) \leq R_m, \tag{15}$$

$$\frac{s_n}{s_n^{'}} + \frac{w_n}{w_n^{'}} + \frac{r_n}{r_n^{'}} \leq d_n, \quad \forall n \in N \cup \{p\}, \tag{16}$$

$$0 \leq s_n^{'} \leq \infty, \quad \forall n \in N \cup \{p\} \tag{17}$$

$$0 \leq w_n^{'} \leq \infty, \quad \forall n \in N \cup \{p\} \tag{18}$$

$$0 \leq r_n^{'} \leq \infty, \quad \forall n \in N \cup \{p\} \tag{19}$$

$$x_n \in \{0, 1\}, \quad \forall n \in N \tag{20}$$

The objective (Eq.(11)) is to maximize the price of all tasks (not including the new task as the price is zero). The server resource capacity constraints are similar to the constraints in the standard model set out in section 3.2 however with the assumption that the task k is running so there is no need to consider if the task is running or not. The deadline and non-negative resource speeds constraints (5, 6, 7 and 8) are all the same equation with the new task included with all of the other tasks. The equation to check that a task is only allocated to a single server is not included as only server i considers the task k's price.

In auction theory, four properties are considered: Incentive compatible, budget balanced, economically efficient and individual rationality.

- Budget balanced - Since the auction is run without an auctioneer, this allows for the auction to be run in a decentralised way resulting in no "middlemen" taking some money so all revenue goes straight to the servers from the tasks
- Individually Rational - As the server need to confirm with the task if it is willing to pay an amount to be allocated, the task can check this against its secret reserved price preventing the task from ever paying more than it is willing
- Incentive Compatible - Misreporting can give a task as if the task can predict the allocation of resources from server to tasks then tasks can misreport so to be allocate to a certain server that otherwise would result in the task being unallocated.
- Economic efficiency - At the begin then task are almost randomly assigned in till server become full and require kicking tasks off, this means that allocation can fall into a local price maxima meaning that the server will sometime not be 100% economically efficient.

Algorithm 2 Decentralised Iterative Auction

Require: I is the set of servers
Require: J is the set of unallocated tasks, which initial is the set of all tasks to be allocated
Require: $P(i, k)$ is solution to the problem in section 4.3.1 using the server i and new task k. The server's current tasks is known to itself and its current revenue from tasks so not passed as arguments.
Require: $R(i, k)$ is a function returning the list of tasks not able to run if task k is allocated to server i
Require: $\leftarrow_R$ will randomly select an element from a set

```
while |J| > 0 do
  j ←_R J
  p, i ← argmin_{i∈I} P(i, j)
  if p ≤ v_j then
    p_j ← p
    x_{i,j} ← 1
    for all j' ∈ R(i, j) do
      x_{i,j'} ← 0
      p'_j ← 0
      J ← J ∪ j'
    end for
  end if
  J ← J \ {j}
end while
```

The algorithm 2 is a centralised version of the decentralised iterative auction. It works through iteratively checking a currently unallocated job to find the price if the job was currently allocated on a server. This is done through first solving the program in section 4.3.1 which calculates the new revenue if the task was forced to be allocated with a price of zero. The task price is equal to the current server revenue − new revenue with the task allocated + a price change variable to increase the revenue of the server. The minimum price returned by $P(i, k)$ is then compared to the job's maximum reserve price (that would be private in the equivalent decentralised algorithm) to confirm if the job is willing to pay at that price. If the job is willing then the job is allocated to the minimum price server and the job price set to the agreed price. However in the process of allocating a job then the currently allocated jobs on the server could be unallocated so these jobs allocation's and price's are reset then appended to the set of unallocated jobs.

4.4 Attributes of proposed algorithms

In table 3, the important attributes for the proposed algorithm

Attribute	GM	CVA	DIA
Truthfulness		Yes	No
Optimality	No	No	No
Scalability	Yes	Yes	No
Information requirements from users	All	All	Not the reserve value
Communication over heads	Low	Low	High
Decentralisation	No	No	Yes

Table 3: Attributes of the proposed algorithms: Greedy mechanism (GM), Critical Value auction(CVA) and Decentralised Iterative auction (DIA)

5 EMPIRICAL EVALUATION

To test the algorithms presented in section 4, synthetic models have been used to generate a list of tasks and servers.

The synthetic models have been handcrafted with each attribute being generated from a gaussian distribution with a mean and standard deviation.

To compare the greedy algorithm to the optimal elastic allocation, a branch and bound was implemented to solve the problem in section 3.2. In order to compare to fixed speed equivalent models, the minimum total resource required to run the job is found and set as the resource speeds for all of the tasks, with the optimal solution for running the job with the fixed speeds is found as well. To implement the greedy mechanism, the value density function was $\frac{v_j}{s_j+w_j+r_j}$, server selection was $argmin_{\forall i \in I} S_i^{'} + W_i^{'} + R_i^{'}$ and the resource allocation was $min s_j^{'} + w_j^{'} + r_j^{'}$ for job j and servers I.

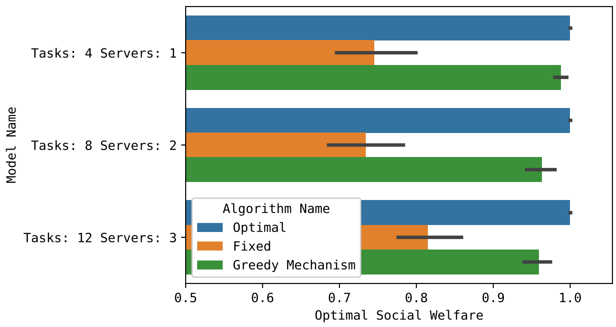


Figure 5: Comparison of the social welfare for the greedy mechanism, optimal, relaxed problem, time limited branch and bound

As figure 5 shows, the greedy mechanism achieves 98% of the optimal solution for the small models, the mechanism achieves within 95% for larger models. In comparison, the fixed allocation achieves 80% of the optimal solution and always does worse than the social welfare of the greedy mechanism.

Figure 6 compares the social welfare of the auction mechanisms: vcg, fixed resource speed vcg, critical value auction and the decentralised iterative auction with different price change variables.

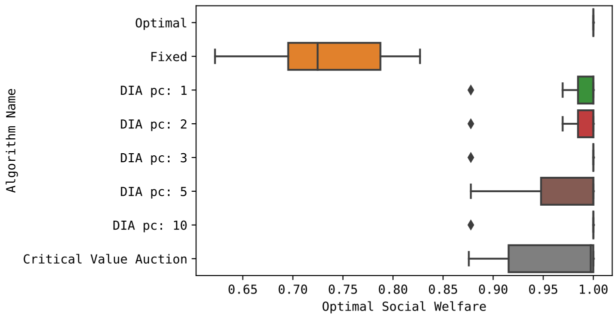


Figure 6: Comparison of the social welfare for the auction mechanisms

VCG is an economically efficient auction that requires the optimal solution to the problem in section 3.2.

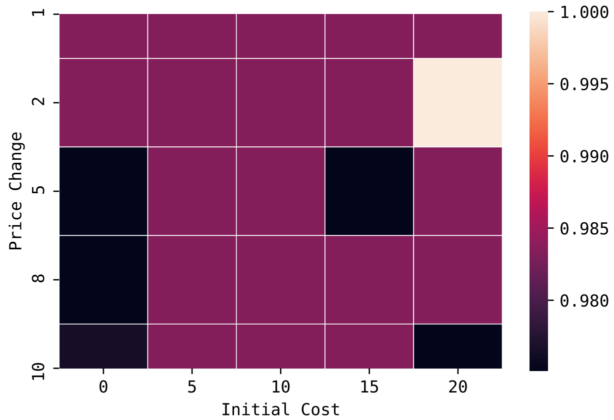


Figure 7: Average number of rounds with a price change variables and task initial cost

Within the context of edge cloud computing, the number of rounds for the decentralised iterative auction is important to making it a feasible auction as it is proportional to the time required to run. We investigated the effect of two heuristic on the number of rounds and social welfare of the auction; the price change variable and initial cost heuristic. With an auction using as minimum heuristic values for the price change and initial cost, figure 7, on average 400 rounds were required for the price to converge while an auction using a price change of 10 and initial cost of 20 means that only on average 80 rounds are required, 5x less. But by using high initial cost and price change heuristics, this can prevent tasks from being allocated, figure 8, shows that the difference in social welfare is only 2% from minimum to maximum heuristics.

6 CONCLUSIONS

In this paper, we studied a resource allocation problem in edge clouds, where resources are elastic and can be allocated to tasks at varying speeds to satisfy heterogeneous requirements and deadlines.

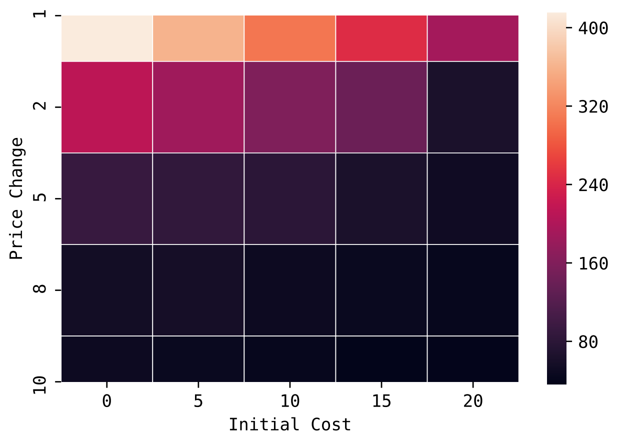


Figure 8: Average social welfare with a price change variables and task initial cost

To solve the problem, we proposed a centralized greedy mechanism with a guaranteed performance bound, and a number of auction-based mechanisms that also consider the elasticity of resources and limit the potential for strategic manipulation. We show that explicitly taking advantage of resource elasticity leads to significantly better performance than current approaches that assume fixed resources.

In future work, we plan to consider the dynamic scenario where tasks arrive and depart from the system over time, and to also consider the case where task preemption is allowed.

REFERENCES

[1] Zubaida Alazawi, Omar Alani, Mohmmad B. Abdljabar, Saleh Altowaijri, and Rashid Mehmood. 2014. A Smart Disaster Management System for Future Cities. In *Proceedings of the 2014 ACM International Workshop on Wireless and Mobile Technologies for Smart Cities (WiMobCity '14)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/2633661.2633670

[2] M. Bahrami. 2015. Cloud Computing for Emerging Mobile Cloud Apps. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. 4–5. https://doi.org/10.1109/MobileCloud.2015.40

[3] Fan Bi, Sebastian Stein, Enrico Gerding, Nick Jennings, and Thomas La Porta. 2019. A truthful online mechanism for resource allocation in fog computing. In *PRICAI 2019: Trends in Artificial Intelligence. PRICAI 2019*, A. Nayak and A. Sharma (Eds.), Vol. 11672. Springer, Cham, 363–376. https://eprints.soton.ac.uk/431819/

[4] Zhiyi Huang Bingqian Du, Chuan Wu. 2019. Learning Resource Allocation and Pricing for Cloud Profit Maximization. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*. 7570–7577.

[5] Edward H. Clarke. 1971. Multipart pricing of public goods. *Public Choice* 11, 1 (01 Sep 1971), 17–33. https://doi.org/10.1007/BF01726210

[6] P. Corcoran and S. K. Datta. 2016. Mobile-Edge Computing and the Internet of Things for Consumers: Extending cloud computing and services to the edge of the network. *IEEE Consumer Electronics Magazine* 5, 4 (2016).

[7] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, and K. S. Chan. 2019. Service Placement and Request Scheduling for Data-intensive Applications in Edge Clouds. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 1279–1287. https://doi.org/10.1109/INFOCOM.2019.8737368

[8] Theodore Groves. 1973. Incentives in Teams. *Econometrica* 41, 4 (1973), 617–631. http://www.jstor.org/stable/1914085

[9] L. Guerdan, O. Apperson, and P. Calyam. 2017. Augmented Resource Allocation Framework for Disaster Response Coordination in Mobile Cloud Environments. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*.

[10] Hans Kellere, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack problems*. Springer.

[11] Dinesh Kumar, Gaurav Baranwal, Zahid Raza, and Deo Prakash Vidyarthi. 2017. A systematic study of double auction mechanisms in cloud computing. *Journal of Systems and Software* 125 (2017), 234 – 255. https://doi.org/10.1016/j.jss.2016.12.009

[12] Y. Liu, F. R. Yu, X. Li, H. Ji, and V. C. M. Leung. 2018. Distributed Resource Allocation and Computation Offloading in Fog and Cloud Networks With Non-Orthogonal Multiple Access. *IEEE Transactions on Vehicular Technology* 67, 12 (2018).

[13] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. 2017. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys Tutorials* 19, 4 (2017).

[14] Kabrane Mustapha, Krit Salah-ddine, and L. Elmaimouni. 2018. Smart Cities: Study and Comparison of Traffic Light Optimization in Modern Urban Areas Using Artificial Intelligence. *International Journal of Advanced Research in Computer Science and Software Engineering* 8 (02 2018), 2277–128. https://doi.org/10.23956/ijarcsse.v8i2.570

[15] Kameng Nip, Zhenbo Wang, and Zizhuo Wang. 2017. Knapsack with variable weights satisfying linear constraints. *Journal of Global Optimization* 69, 3 (01 Nov 2017), 713–725. https://doi.org/10.1007/s10898-017-0540-y

[16] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. 2007. *Algorithmic game theory*. Cambridge university press. 229 pages. https://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf

[17] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. 2007. *Algorithmic game theory*. Cambridge university press. 229–230 pages. https://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf

[18] Mallesh M. Pai and Aaron Roth. 2013. Privacy and Mechanism Design. *SIGecom Exch.* 12, 1 (June 2013), 8–29. https://doi.org/10.1145/2509013.2509016

[19] M. Sapienza, E. Guardo, M. Cavallo, G. La Torre, G. Leombruno, and O. Tomarchio. 2016. Solving Critical Events through Mobile Edge Computing: An Approach for Smart Cities. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*.

[20] G. Sreenu and M. A. Saleem Durai. 2019. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data* 6, 1 (06 Jun 2019), 48. https://doi.org/10.1186/s40537-019-0212-5

[21] William Vickrey. 1961. Counterspeculation, Auctions, and Competitive Sealed Tenders. *The Journal of Finance* 16, 1 (1961), 8–37. http://www.jstor.org/stable/2977633

[22] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. M. Lau. 2017. Online Auctions in IaaS Clouds: Welfare and Profit Maximization With Server Costs. *IEEE/ACM Transactions on Networking* 25, 2 (April 2017), 1034–1047. https://doi.org/10.1109/TNET.2016.2619743

References

tocchapterAppendix References

M. Bahrami . Cloud computing for emerging mobile cloud apps. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 4–5, March 2015. .

V. Farhadi , F. Mehmeti , T. He , T. L. Porta , H. Khamfroush , S. Wang , and K. S. Chan . Service placement and request scheduling for data-intensive applications in edge clouds. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1279–1287, April 2019. . URL https://ieeexplore.ieee.org/document/8737368.

Y. Mao , C. You , J. Zhang , K. Huang , and K. B. Letaief . A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys Tutorials*, 19(4), 2017.

Zubaida Alazawi, Omar Alani, Mohmmad B. Abdljabar, Saleh Altowaijri, and Rashid Mehmood. A smart disaster management system for future cities. In *Proceedings of the 2014 ACM International Workshop on Wireless and Mobile Technologies for Smart Cities*, WiMobCity '14, pages 1–10, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3036-7. . URL http://doi.acm.org/10.1145/2633661.2633670.

Fan Bi, Sebastian Stein, Enrico Gerding, Nick Jennings, and Thomas La Porta. A truthful online mechanism for resource allocation in fog computing. In A. Nayak and A. Sharma, editors, *PRICAI 2019: Trends in Artificial Intelligence. PRICAI 2019*, volume 11672, pages 363–376. Springer, Cham, August 2019. URL https://eprints.soton.ac.uk/431819/.

Zhiyi Huang Bingqian Du, Chuan Wu. Learning resource allocation and pricing for cloud profit maximization. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, pages 7570–7577, 2019.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48, 2001.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. . URL https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, October 2016. ISSN 00280836. URL http://dx.doi.org/10.1038/nature20101.

S.R. Gunn. Pdflatex instructions, 2001. URL http://www.ecs.soton.ac.uk/~srg/softwaretools/document/.

S.R. Gunn and C. J. Lovell. Updated templates reference 2, 2011.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. . URL https://doi.org/10.1162/neco.1997.9.8.1735.

Dinesh Kumar, Gaurav Baranwal, Zahid Raza, and Deo Prakash Vidyarthi. A systematic study of double auction mechanisms in cloud computing. *Journal of Systems and Software*, 125:234 – 255, 2017. ISSN 0164-1212. . URL http://www.sciencedirect.com/science/article/pii/S0164121216302540.

C. J. Lovell. Updated templates, 2011.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. . URL https://doi.org/10.1007/BF02478259.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Kabrane Mustapha, Krit Salah-ddine, and L. Elmaimouni. Smart cities: Study and comparison of traffic light optimization in modern urban areas using artificial intelligence. *International Journal of Advanced Research in Computer Science and Software Engineering*, 8:2277–128, 02 2018. .

Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic game theory*. Cambridge university press, 2007. URL https://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf.

OpenAI. Openai five. https://blog.openai.com/openai-five/.

Mallesh M. Pai and Aaron Roth. Privacy and mechanism design. *SIGecom Exch.*, 12(1):8–29, June 2013. ISSN 1551-9031. . URL http://doi.acm.org/10.1145/2509013.2509016.

Arthur L Samuel. Some studies in machine learning using the game of checkers. ii—recent progress. In *Computer Games I*, pages 366–400. Springer, 1988.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017. URL http://dx.doi.org/10.1038/nature24270.

G. Sreenu and M. A. Saleem Durai. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data*, 6(1):48, Jun 2019. ISSN 2196-1115. . URL https://doi.org/10.1186/s40537-019-0212-5.

Mark Towers, Sebastian Stein, Fidan Mehmeti, Caroline Rubeun, Tim Norman, Tom La Porta, and Geeth Demel. Auction-based mechanisms for allocating elastic resources in edge clouds. Unpublished.

William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2977633.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017. URL http://arxiv.org/abs/1708.04782.

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*, 25(2):1034–1047, April 2017. .

Bibliography

P. Corcoran and S. K. Datta . Mobile-edge computing and the internet of things for consumers: Extending cloud computing and services to the edge of the network. *IEEE Consumer Electronics Magazine*, 5(4), 2016.

V. Farhadi , F. Mehmeti , T. He , T. L. Porta , H. Khamfroush , S. Wang , and K. S. Chan . Service placement and request scheduling for data-intensive applications in edge clouds. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1279–1287, April 2019. . URL https://ieeexplore.ieee.org/document/8737368.

L. Guerdan , O. Apperson , and P. Calyam . Augmented resource allocation framework for disaster response coordination in mobile cloud environments. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, 2017.

Y. Mao , C. You , J. Zhang , K. Huang , and K. B. Letaief . A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys Tutorials*, 19(4), 2017.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

May Al-Roomi, Shaikha Al-Ebrahim, Sabika Buqrais, and Imtiaz Ahmad. Cloud computing pricing models: a survey. *International Journal of Grid and Distributed Computing*, 6(5):93–106, 2013.

Zubaida Alazawi, Omar Alani, Mohmmad B. Abdljabar, Saleh Altowaijri, and Rashid Mehmood. A smart disaster management system for future cities. In *Proceedings of the 2014 ACM International Workshop on Wireless and Mobile Technologies for Smart Cities*, WiMobCity '14, pages 1–10, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3036-7. . URL http://doi.acm.org/10.1145/2633661.2633670.

Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy P. Lillicrap. Distributed distributional deterministic policy gradients. *CoRR*, abs/1804.08617, 2018. URL http://arxiv.org/abs/1804.08617.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017. URL http://arxiv.org/abs/1707.06887.

Fan Bi, Sebastian Stein, Enrico Gerding, Nick Jennings, and Thomas La Porta. A truthful online mechanism for resource allocation in fog computing. In A. Nayak and A. Sharma, editors, *PRICAI 2019: Trends in Artificial Intelligence. PRICAI 2019*, volume 11672, pages 363–376. Springer, Cham, August 2019. URL https://eprints.soton.ac.uk/431819/.

Zhiyi Huang Bingqian Du, Chuan Wu. Learning resource allocation and pricing for cloud profit maximization. In *The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, pages 7570–7577, 2019.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 11(1):17–33, Sep 1971. ISSN 1573-7101. . URL https://doi.org/10.1007/BF01726210.

Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48, 2001.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. . URL https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. *CoRR*, abs/1706.10295, 2017. URL http://arxiv.org/abs/1706.10295.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018. URL http://arxiv.org/abs/1802.09477.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, October 2016. ISSN 00280836. URL http://dx.doi.org/10.1038/nature20101.

Theodore Groves. Incentives in teams. *Econometrica*, 41(4):617–631, 1973. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1914085.

S.R. Gunn. Pdflatex instructions, 2001. URL http://www.ecs.soton.ac.uk/~srg/softwaretools/document/.

S.R. Gunn and C. J. Lovell. Updated templates reference 2, 2011.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017. URL http://arxiv.org/abs/1710.02298.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. . URL https://doi.org/10.1162/neco.1997.9.8.1735.

Dinesh Kumar, Gaurav Baranwal, Zahid Raza, and Deo Prakash Vidyarthi. A systematic study of double auction mechanisms in cloud computing. *Journal of Systems and Software*, 125:234 – 255, 2017. ISSN 0164-1212. . URL http://www.sciencedirect.com/science/article/pii/S0164121216302540.

C. J. Lovell. Updated templates, 2011.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. . URL https://doi.org/10.1007/BF02478259.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

Kabrane Mustapha, Krit Salah-ddine, and L. Elmaimouni. Smart cities: Study and comparison of traffic light optimization in modern urban areas using artificial intelligence. *International Journal of Advanced Research in Computer Science and Software Engineering*, 8:2277–128, 02 2018. .

Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic game theory*. Cambridge university press, 2007. URL https://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf.

OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015. URL http://arxiv.org/abs/1511.05952. cite arxiv:1511.05952Comment: Published at ICLR 2016.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/silver14.html.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017. URL http://dx.doi.org/10.1038/nature24270.

G. Sreenu and M. A. Saleem Durai. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data*, 6(1):48, Jun 2019. ISSN 2196-1115. . URL https://doi.org/10.1186/s40537-019-0212-5.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3(1):9–44, August 1988. ISSN 0885-6125. . URL https://doi.org/10.1023/A:1022633531479.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Mark Towers, Sebastian Stein, Fidan Mehmeti, Caroline Rubeun, Tim Norman, Tom La Porta, and Geeth Demel. Auction-based mechanisms for allocating elastic resources in edge clouds. Unpublished, 2020.

Alan M Turing. Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer, 2009.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2977633.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017. URL `http://arxiv.org/abs/1708.04782`.

Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015. URL `http://arxiv.org/abs/1511.06581`.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8 (3-4):279–292, 1992.

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*, 25(2):1034–1047, April 2017. .