Auction-based Mechanisms for Resource Elastic Tasks in Edge Cloud Computing

MARK TOWERS and SEBASTIAN STEIN, University of Southampton, United Kingdom
FIDAN MEHMETI and TOM LA PORTA, Penn State University, United States
GEETH DE MEL, IBM UK, United Kingdom

Edge Cloud Computing enables computational tasks to be process at the edge of the network using limited computational resource in comparison to larger remote data centres. Because of this, resource allocation and management is significantly more important. Existing resource allocations approaches usually assumed that tasks have inelastic resource requirements (i.e., a fixed amount of bandwidth and computation requirements), however, this may result in inefficient resource usage due to unbalanced requirements from tasks that can result in bottlenecks. To address this, we propose a novel approach that takes advantage of the elastic nature of resources, This due to time taken for an operation to occur is generally proportional to the resource allocated. This, however, makes previous research incompatible with such elasticity. Therefore, we describe this problem formally as an optimisation problem and propose a scalable approximation algorithm. To deal with self-interested users, we demonstrate a centralised auction mechanism that is incentive compatible using the approximation algorithm. Moreover, we propose a novel Decentralised Iterative Auction that does not require users to reveal their private task value. Using extensive simulations, we show that considering the elasticity of resources leads to a gain in utility of around 20% compared to existing approaches, with the proposed approaches typically achieving 95% of the theoretical optimal.

Additional Key Words and Phrases: Edge clouds; elastic resources; auctions

ACM Reference Format:
Mark Towers, Sebastian Stein, Fidan Mehmeti, Tom La Porta, and Geeth De Mel. 2020. Auction-based Mechanisms for Resource Elastic Tasks in Edge Cloud Computing. 1, 1 (July 2020), 23 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the last few years, Cloud Computing [2] has become a popular solution for running data-intensive applications remotely. However, large-scale data centres are not feasible for application domains that require low latency or better application performance [6]. To deal with such domains, *Fog/Edge Computing* has emerged as a complementary paradigm allowing tasks to be executed at the edge of networks, close to the user, in small data centers, known as *edge nodes*.

As the Internet-of-things grows, Fog/Edge Cloud Computing is a key enabling technology [6, 25], in particular for applications in smart cities [26], disaster response scenarios [1, 11], home automation systems, etc. In these applications, low-powered devices generate data or tasks that cannot be processed locally but are impractical to use with standard

Authors' addresses: Mark Towers, mt5g17@soton.ac.uk; Sebastian Stein, ss2@soton.ac.uk, University of Southampton, Southampton, United Kingdom; Fidan Mehmeti, fzm82@psu.edu; Tom La Porta, tfl12@psu.edu, Penn State University, Pennsylvania, United States; Geeth De Mel, geeth.demel@uk.ibm.com, IBM UK, United Kingdom.

Manuscript submitted to ACM

cloud computing services. More specifically, in smart cities, these devices could be used to control smart traffic light systems [18] which collect data from roadside sensors to optimise the traffic light sequence, minimising vehicle waiting times, or to analyse video feeds from CCTV cameras [27]. In disaster response [1], sensor data from autonomous vehicles can be aggregated to produce real-time maps of devastated areas to help first responders better understand the situation and search for survivors.

To accomplish these tasks, there are typically several types of resources that are needed, including but not exclusively communication bandwidth, computational power, and data storage resources [8]. These are tasks are generally delay-sensitive, meaning the task may be finished as fast as possible or by a specific completion deadline.
When accomplished, different tasks carry different values for their owners often dependant on the importance of the program tasks, 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 criminal on the run.
Given that edge clouds are often highly constrained in their resources [15], 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, since 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 [23].

An important shortcoming of existing work around resource allocation in edge cloud computing is that it assumes tasks have strict resource requirements – that is, each task must be allocated a fixed amount of CPU cycles or bandwidth by a server. This is often achieved by users selecting a particular VM specification for their task, however, this can cause both inefficient allocation of resources and bottlenecks on certain server resources when multiple tasks over-request particular resources.

Previous work [12] has considered the ability of cloud servers to flexibly respond to usage by expanding server capacity when needed. Due to the often ad hoc nature of fog cloud networks and that cloud nodes don't exist in a centralised location but often across the global, such an ability is generally not feasible. Therefore in this work we instead utilise the ability for tasks to have flexibility (not server) in how its resources are allocated due to the linear relationships of many task attributes.

An example of elasticity is the time taken for a program to be download by the server, this is generally proportional to the bandwidth allocated. It is similarly true for sending back of results to the users. Computation is more difficult as the scalability of a program is dependant on the particular program, therefore this work considers only tasks that are linearly scalable. We leave for tasks to be able to scale non-linearly for future work.

Using this capability to elastically allocate resources is additionally important in the case of edge computing due to the limited resource capacities that servers have in comparison to the large data centres used in standard cloud computing. Therefore, using resource elasticity, we propose a novel resources allocation optimisation problem that enables servers to have greater control over their resource allocation. However, this optimisation problem also incompatible with previous research in this area, thus we propose three mechanisms that utilise this flexibility. These are a modular greedy algorithm, a centralised incentive-compatible auction, and a novel Decentralised Iterative Auction.

In the following section, an overview of related work is provided (Section 2). Section 3 presents the problem formulation, a formal optimisation problem, and an example case to show the effectiveness elastic resource allocation compared to fixed resource allocation. Using the developed formulation, Section 4 presents our three algorithms. Using these algorithms, Section 5 presents an extensive analysis of our algorithms.

2 RELATED WORK

Within Fog Computing, there exist a wide range of approaches to resource management including: application placement, resource scheduling, task offloading, load balancing, resource allocation, and resource provisioning [9]. As this work bridges a range of these topics, an outline of related works from several approaches have been explained.

Auctions are a popular method for both dealing with self-interested users and in determining the value when allocating multiple limited resources.

Due to the large corpus of research, Kumar et al. provides a systematic study of double auction mechanisms that are suitable for a range of distributed systems like Grid computing, Cloud computing and Inter-Cloud systems [14]. The work reviewed 21 different proposed auction mechanisms over a range of important auction properties like Economic Efficiency, Incentive Compatibility, and Budget-Balance. In a majority of the proposed auction mechanisms, truthfulness was only considered for the user, thus a Truthful Multi-Unit Double auction mechanism was presented as such that both users and servers should act truthfully.

Edge-MAP [28] provides a client-to-cloud model for tasks with extremely short deadlines (10-50ms). Through using a Vickrey-English-Dutch (VED) auction, the system arrives at the unique minimum competitive equilibrium price. Because of this, the system is highly scalable and adaptable to dynamically changing network topologies.

An alternative market-based framework by Nguyen et al., allows for resources to be efficiently allocated by edge nodes that are geographically distributed [19]. To maximise the use of node resource, the framework finds the market equilibrium through optimally allocating resources bundles to services given each task's budget. The market's equilibrium is found using the Eisenberg-Gale convex program which allows services to maximise their revenue. They additionally proved a novel convex optimisation problem that achieves market equilibrium when services instead maximise their net profit (revenue minus cost).

An advantage of fog nodes is that due to the proximity, nodes can find other nodes in its vicinity to assist in a task [17]. Using this ability, resource allocation mechanisms have been proposed to allow fog nodes to offload multiple tasks with delay guarantees. This is helpful due to the limited resources of nodes, this allows a node to offload a task to either other fog nodes or remote cloud centers for further computational help. However this causes an issue in deciding where to offload and how much partial task data to be offloaded under delay guarantee. By formulating the problem as a Quadratically Constraint Quadratic Programming allows for a solution to be found.

Quality of service is an alternative metric for measuring the success of a system compared to social welfare used in a majority of mechanisms. Chen et al. propose a resource-efficient computation offloading mechanism for users, where communication and computation are joined by the node operation when allocating resources [4]. Using a task graph model, the resource demands of a task are calculated, which are used to compute the optimal communication and computation resource demand profile for a user in order to minimise a tasks resource occupancy. As the problem is a NP-Hard problem, through an efficient approximation algorithm, a truthful pricing scheme is used to calculate the critical value for the task to prevent users from misreporting task valuation.

As Fog Computing often requiring the placement of fog nodes within the system for both localised fog networks and geographically distance systems due to the short deadline constraints of tasks. Work by Farhadi et al., considers the placement of code/data needed to run specific tasks over time where the demands change over time while also considering the operational costs and system stability [8]. Using a proposed approximation algorithm, they achieved 90% of the optimal social welfare by converting the problem to a set function optimisation problem. By doing this, this allows the algorithm to run in polynomial time complexity.

Alternative work aimed to efficiently distribute data centers and connections aims to maximise social welfare instead of the number of tasks completed [3]. A truthful online mechanism was proposed that was incentive compatible and individually rational, to allow tasks to arrive over time by solving an integer programming optimisation problem.

All of the approaches explained above for task pricing and resource allocation in Cloud Computing use a form of fixed resource allocation, where each user requests a fixed amount of resources for a task from a server. This is often achieved through offering users a range of VM of different specifications for tasks to use. However, this mechanism, as previously explained, provides no control for the server over the amount of resources allocated to a task, only allowing the server to determine a task's price.

As our problem is related to multidimensional knapsack problems where there is a large body of works [7, 16]. Very little work has been done to allow for flexibility of item weights with linear constraint that must be fulfilled for the item to be allocated [20]. The work provides a pseudo-polynomial time complexity algorithm for solving this problem to maximize the values in the knapsack. While our optimisation problem case is similar to this work, it differs due to the constraints with our using non-linear constraints making it unusable.

3 PROBLEM FORMULATION

In this section, an outline of the system model describes servers and tasks attributes (subsection 3.1) used in the resource elastic optimisation problem (subsection 3.2). Using this model, we prove that it is NP-Hard (subsection 3.3). Finally, using a example program, we demonstrate in the effectiveness of our flexible resource allocation scheme compared to an fixed resource allocation scheme used in previous work, as explained the previous section (subsection 3.4).

3.1 System model

A sketch of the system is shown in Fig. 1. We assume that there are a set of servers $I = \{1, 2, \ldots, |I|\}$, which could be accessed either through cellular base stations or WiFi access points (APs). These servers have different types of limited 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 Ghz), and communication bandwidth to receive the data and to send back the results of the task after execution per time interval (e.g., measured in Mbit). The servers are assumed to only consider these attributes for resource allocation, however future work would wish to explore additional attributes like I/O memory access per second, GPU usage, and more. The servers are also assumed to be heterogeneous in all their characteristics. Formally, we denote the storage capacity of server i with S_i, the computation capacity with W_i, and the bandwidth capacity with R_i.

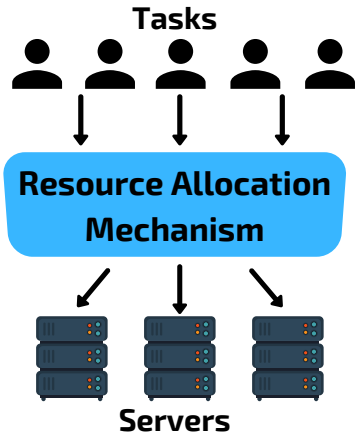

Fig. 1. System Model

The system also contains a set $J = \{1, 2, \ldots, |J|\}$, of different tasks that each require service from one of the servers I. Each task has a monetary value, denoted v_j, representing the maximum price the owner is willing to pay for the task to be computed. In order to run a task, the server is required to load the appropriate code/data onto the server from a source, then to compute the code of the task and to send back results to the user. Therefore, for each of these stages, we consider

separate speeds that the operations could occur at to enable the greatest flexibility for the server at each stage. The storage size for the task j is denoted as s_j with the rate that the program is transferred to the server as $s_j^{'}$. 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 number of CPU cycles assigned to the task per unit of time as $w_j^{'}$. 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 bandwidth used to sent back results to the user as $r_j^{'}$.

In order to force the server to complete the task with a reasonable time, each task sets a deadline, denoted by d_j, representing the maximum amount of time for a task to be completed successfully within. This includes: the time required to load the data/code onto the server, run it, and send back results to the user. We assume that there is an *all*-or-*nothing* task execution reward scheme, meaning that the task value is awarded only if the task is completed within its deadline.

3.2 Optimisation problem

Given the aforementioned assumptions and variables from the system model, an optimisation problem is constructed as followed. This problem uses the additional variable $x_{i,j}$ to denote the allocation of a task j that will run on server i.

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

s.t.

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

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

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

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

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

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

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

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

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

The objective (eq. 1) is to maximize the total value over all tasks (i.e., social welfare) that are completed within their deadline (eq. 5). Constraints 2, 3 and 4, prevent over allocation of server resources to allocated tasks. For the server's storage capacity (constraint 2), each server's storage must be less than the allocated task's storage requirements. While for the server's computational capacity (constraint 3), the capacity is limited by a server's allocated tasks compute resources ($w_j^{'}$). The server's bandwidth capacity (constraint 4) comprises of two parts: the first for loading of data/code of a task onto the server and the second for sending back result to the user.

To force the task to the completed within its assigned deadline, constraint 5 required the sum of time taken for each stages of the task, completed in series, to be less than the deadline value. Note that if a task is not allocated to any server, this constraints can be satisfied by choosing arbitrarily resource speed as these resources do not use up any servers' resources in constraint 2, 3 or 4.
Constraints 6, 7, 8 enforce that the resource speeds for each stage ($s_j^{'}$, $w_j^{'}$, and $r_j^{'}$) are all positive and finite. Finally, as every task can only be served by at most one server, constraints 9 and 10 enforce this.

This model focus on a single-shot setting where all tasks arrival at the same time to the system. To use this system in practice where tasks arrival progressively over time, an allocation mechanism would repeat the allocation decisions described here over regular time intervals. As a result, longer running tasks would reappearing in consecutive time intervals. In subsection 5.6, we evaluate the effectiveness of such a batching mechanism compared to online mechanisms. We leave a detailed study of online mechanisms to future work. A probably advantage of such a system is the ability to dynamically change the resource allocation at each time step.

3.3 Time Complexity

As the optimisation problem as described in Subsection 3.2 is an extension of the Knapsack problem, a well-studied problem in computer science that known to be NP-Hard. By transforming the problem into a standard knapsack problem, the time complexity of the problem is also NP-Hard.

Theorem 3.1. *The optimisation problem in subsection 3.2 is NP-hard.*

Proof. The task resource elasticity 5 can removed from the optimisation problem to simplify the model by setting the task resource speeds to a fixed value that satisfies the deadline constraint. This reduces the model to a 0–1 multidimensional knapsack problem [13], which is a generalization of a simple 0–1 knapsack problem. The latter is an NP-hard problem [13]. Given this, it follows that the 0–1 multidimensional knapsack problem is also NP-hard. Since the optimization problem (Eqs. 1 - 10) is a generalization of a 0–1 multidimensional knapsack problem, it follows that it is NP-hard as well. □

3.4 Example Problem Case

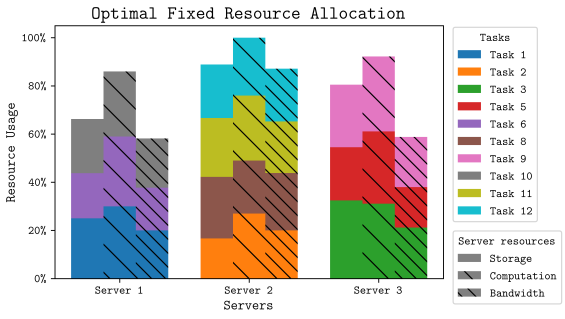

Fig. 2. Optimal solution with fixed resource speeds

Before we propose our allocation mechanisms in the next section, we present an example case to illustrate why elasticity is important. In this example, there are 12 potential tasks and 3 servers where the flexible solution is able to achieve 18% better social welfare compared to the fixed resource allocation solution. The exact settings can be found in Appendix A with tables ?? for the task and server attributes.
The figures 2 and 3 represent each server as a group of three bars, each relating to each server's resource type, with the percentage of resources used by a task being the size of the bar.

Figure 2 shows the best possible allocation if tasks have fixed resource speeds (which were set by minimising the total amount of resources to be completed within the

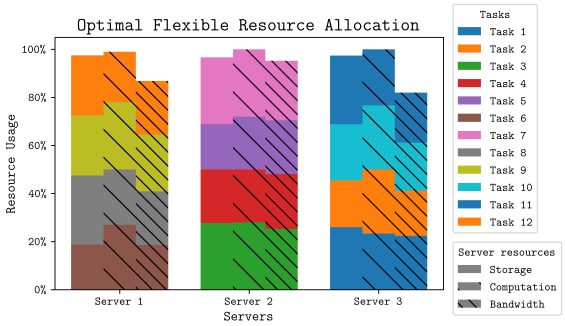

Fig. 3. Optimal solution with elastic resources speeds

deadline). Here, only 9 of the tasks are run, resulting in a total social welfare of 980 due to server 1 and 2's limited computational capacity and server 3' limited communication capacity.

In contrast to figure 2, Figure 3 depicts the optimal allocation if elastic resources are considered. Here, all the resources are used by the servers, whereas the fixed example 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 18% improvement over the fixed approach).

4 FLEXIBLE RESOURCE ALLOCATION MECHANISMS

As explained in the last Section, all previous research outlined in Section 2 is incompatible with our resource elastic optimisation problem in Section 3. Therefore in this section, we propose three mechanisms for solving this optimisation problem: an approximation algorithm and two auction-based mechanisms.

The optimisation problem is a extended version of the knapsack problem, which is often solved using an dynamic programming method that has pseudo-polynomial time complexity [29]. The solution requires building a table of items to bags allocation. For our problem, as the resource speeds must be considered at the same time, such a solver can are infeasible due to both the space and time complexity required.

Because of this issue of allocating both tasks to a server and a server's resources to a task, this work proposes an approximation algorithm where tasks are allocated to a server with resources in series not in parallel. The centralised greedy algorithm (detailed in Subsection 4.1) ranks tasks that are each allocated to a server with resources that at each stage uses a separate ranking function. This algorithm has a social welfare lower bound of $\frac{1}{|J|}$, however in practice achieves close to the theoretical optimal while running with polynomial time complexity.

As task users can be self-interested, they may report their task values or requirements strategically. Traditionally, VCG [5, 10, 30] is used for such systems due to its ability to use any optimisation problem to calculate a task price. However, due to the difficulty of calculating the optimal allocation for this problem, VCG is infeasible to use in this application. Therefore, to deal with self-interested users, we propose two auction-based mechanisms. The first being an incentive-compatible auction using the centralised greedy algorithm (Section 4.2) and the second being a novel Decentralized Iterative Auction (Section 4.3) that does not require users to reveal the task value.

4.1 Greedy Algorithm

To solve a knapsack problem, a greedy approximation algorithm is often used [24]. We have applied a similar approach to this problem specified in subsection 3.2. As a result of the elastic nature of task resources, an additional stage is required once a task has been allocated to a server in order to determine the task's resource speeds.

More specifically, the greedy algorithm (algorithm 1) has two stages; stage one sorts the list of tasks based on the value density of each task calculated using task attributes: value, required resources and deadline. The second stage uses the sorted list of tasks to iterate through applying two heuristics to select the server based on available server resources and to allocate resources based on the available server resources and the required resources of the task.

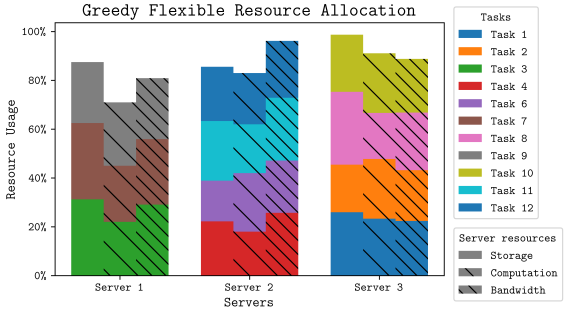

Fig. 4. Example Greedy allocation using the model from table ?? with task and server attributes

Using the same example case from subsection 3.4, the greedy algorithm can complete 11/12 of the tasks achieving XX% more social welfare than the fixed solution. This is due to the algorithm being unable to consider other tasks resource requirement while allocating resources. The greedy algorithm uses the value density function:

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

server selection:

$$\text{argmin}_{\forall i \in I} S_i^{'} \cdot W_i^{'} \cdot R_i^{'}$$

for task j and servers I and the resource allocation:

$$\text{argmin}_{s_j^{'}, w_j^{'}, r_j^{'}} \left(\frac{w_j^{'}}{W_i^{'}}\right)^3 + \left(\frac{s_j^{'} + r_j^{'}}{R_i^{'}}\right)^3$$

is task j and server i.

4.1.1 Greedy Lower Bound. The lower bound of the algorithm is $\frac{1}{|J|}$ (where $|J|$ is the number of tasks) with the task value as the value density function. This lower bound is not affected by the server selection policy or the resource allocation policy.

However in testing, we found that the task value function is not the best value density heuristic as it does not consider the effect of deadlines or the required resources of the task. In Section 5, we considered a wide range of heuristics, showing the results of the best heuristics over a range of settings.

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

Proof. Due to a task not considering other task's resource requirements when allocating resources then no matter the server selection or resource allocation function, it can't be guaranteed that subsequent tasks can be allocated to any server. As a result, the algorithm can be guaranteed to achieve at least $\frac{1}{n}$ of the optimal social welfare by sorting the

Algorithm 1 Pseudo code of Greedy Algorithm

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) of server i
Require: $v(j)$ is the value density function of task j
Require: $s(j, I)$ is the server selection function of task j and set of servers I returning the best server, or $\emptyset$ if the task is not able to be run on any server
Require: $r(j, i)$ is the resource allocation function of a task and server returning a tuple of 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 X and a function for comparing elements f

```
J' ← sort(J, v)
for all j ∈ J' do
    i ← s(j, I)
    if i ≠ ∅ then
        s'_j, w'_j, r'_j ← γ(j, i)
        x_{i,j} ← 1
    end if
end for
```

tasks by task value ($v(j) = j_v$). Using this, the first task from the sorted task list will have the maximum task value meaning the lower bound of the algorithm is $\frac{1}{n}$ of the optimal social welfare. □

4.1.2 Greedy Time Complexity. Using the greedy algorithm (algorithm 1), the time complexity is polynomial, $O(|J|\,|I|)$.

Theorem 4.2. *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 stage 1 of the algorithm is $O(|J|\log(|J|))$ due to sorting the tasks and stage 2 has complexity $O(|J|\,|I|)$ due to looping over all 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 algorithm was used to allocate resources such that the value is the price paid. This would be 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 strategy-proof (weakly dominant incentive compatible) so users have no incentive to misreport task attributes.

Single-Parameter domain auctions are extensively studied in mechanism design [21] and are used where an agent's valuation function can be represented as a single value. The task price is calculated by finding the critical value, the minimum task price required for the task still allocated to any server. This has been shown to be a strategy-proof [22] auction making it a weakly dominant strategy for all users to honestly reveal their task attributes.

The auction (specified in algorithm 2) is implemented using the greedy algorithm from subsection 4.1 both in finding the initial allocation and critical value for each task (at least in a modified form). This helpful as it allows the auction to inherit many of the properties of the greedy algorithm like social welfare and time complexity.
By running the greedy algorithm with all of tasks reported values then an initial allocation can be found. Then for

Algorithm 2 Pseudo code of the Critical Value Auction

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) of server i
Require: $v(j)$ is the value density function of task j
Require: v^{-1} is the inverse of the value density function making the value the subject of the function
Require: $s(j, I)$ is the server selection function of task j and set of servers I returning the best server, or $\emptyset$ if the task is not able to be run on any server
Require: $r(j, i)$ is the resource allocation function of a task and server returning a tuple of 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 X and a function for comparing elements f
Require: can_allocate(j, I) determines whether task j can be allocated to any server I
Require: Greedy(J, I, v, s, r) is the greedy algorithm (subsection 4.1) with tasks J, servers I, value density v, server selection policy s, resource allocation policy r.

```
Greedy(J, I, v, s, r)
for all j' ∈ {j ∈ J|∃x_{j,i} ∀i ∈ I} do
  for all j ∈ J do
    if j' ≠ j then
      i ← s(j, I)
      if i ≠ ∅ then
        s'_j, w'_j, r'_j ← γ(j, i)
      end if
      if ¬can_allocate(j', I) then
        p_{j'} ← v^{-1}(v(j), j')
      end if
    end if
  end for
end for
```

each task that was allocated the critical value for the task must be found meaning that the task will pay the minimum possible value (price) such that the task would still have been allocated by the greedy mechanism.

To find the critical value of the task (referred to here as the critical task) is relatively simple by just modifying the greedy algorithm. So when looping through the list of tasks sorted by value density, the critical task is ignored here, once a task is allocated to a server with resources a check is done to see if the critical task could be allocated to any server. This check has linear time complexity as it is assumed that the server could allocate all of its available resources to the task, only requiring the bandwidth resources to split between loading and sending speeds. Therefore a linear search is required to check if any split would allow the task to be completed.

This process of allocating tasks and checking if the critical task could be allocated is repeated till the critical task can't be allocated to any server. At which point, the value density of the task just allocated is equal to the minimum value density of the critical task such that the critical task could have been allocated. Using this information, the critical value is equal to the inverse of the value density function (where the value is the subject of the function not the value density) using the value density of the previously allocated task's value density.

4.2.1 Critical Value Auction Time Complexity. The time complexity of the auction is $O(|J|\,|J|\,|I|)$, the greedy algorithm repeated $|J|$ due to calculating each task's critical value.

Theorem 4.3. *The time complexity of the critical value auction is* $O(|J|\,|J|\,|I|)$.

Proof. As explained more fully in subsection 4.2, the auction uses the greedy algorithm (subsection 4.1) that has time complexity $O(|J||I|)$. The auction then repeats a modified version of the greedy algorithm for each task allocated initially such that after each task is allocated a check is done to see if the current task could be allocated to any server. This is done by repeating the server selection and resource allocation functions for each task (excluding the critical task) with time complexity, $O(|J||I|)$, until the critical task can no longer be allocated to any server (a linear time function). The task's critical value is calculated in constant time function. As a result, the time complexity for calculating the critical value for an individual task is $O(|J||I|)$. Thus, the overall time complexity is $O(|J||J||I|)$ due to the critical value being found for every task. □

4.2.2 Demonstrating that the Critical Value Auction is Strategyproof. In order that the auction is strategyproof, the value density function must be monotonic [22]. This ensures that if a task misreports any attributes, it will result in the value density of the task decreasing, increasing the price payed. Therefore a value density function must be in the form of $\frac{v_j \cdot d_j}{\alpha(s_j, w_j, r_j)}$ such it is monotonic decreasing if a task attribute is misreported.

Theorem 4.4. *The value density function $\frac{v_j \cdot d_j}{\alpha(s_j, w_j, r_j)}$ is monotonic increasing for task j assuming the function $\alpha(s_j, w_j, r_j)$ is monotonic increasing for each variable.*

Proof. In order to misreport the task value and deadline, misreported values must be less than their true value. Therefore if the value or deadline are decreased then the value density will likewise decrease.
The opposite is true for a task's required resources (storage, compute and result data), as the misreported value must be greater than the true value otherwise the task would not be able to be completed. Therefore as the α function will increase as the resource requirements increase, the resulting value density will decrease.
So in all cases, the overall value density will decrease if the owner doesn't accurately report a task's attribute, making the function monotonic. □

4.3 Decentralised Iterative Auction

In some applications of edge cloud computing, keeping the value of a task a secret is important, for example in military-tactical networks. Therefore we propose a novel Decentralised Iterative Auction based on the pricing principle of the VCG auction [5, 10, 30]. VCG auction calculates the price of an item by finding the difference in social welfare if the item exists and when it doesn't exist.

Our proposed novel auction uses the same principle, except in reverse by finding the difference between the current server revenue and the revenue when the task is allocated with a price of zero. To cause the overall revenue and server revenue to increase, a small value called the Price change variable is added to the task price.

Our auction uses this principle by iteratively letting a task advertise its requirements to all the servers, who respond with their price to run the task. This price is equal to the server's current revenue minus the solution to the problem in subsection 4.3.1 plus the price change variable. This is done to ensure that the total server revenue increases by accepting the task. Once all of the servers have responded, the task can compare the minimum server prices to its private evaluation. This allows the task to privately select the server that it runs on (assumed to be the server with the lowest price) otherwise the task will stop advertising as it knows that its evaluation is lower than the price of any server.

An additional heuristic is used as well to reduce the number of rounds required to find the revenue optima at setting an initial price for a task. This is useful to account for the minimum costs of running the task. The heuristic is referred to as the Initial Price which is set individually by each server.

The algorithm 3 is a centralised version of the auction. The auction runs iteratively till no task is left so either the task can't be allocated to any server or is allocated to a server for a price. In this centralised version, a random task is selected which is then advertised to all of the server who each solve the problem in subsection 4.3.1 to calculate the task price. The minimum price returned by $P(i,k)$ is compared to the task's reverse price (the maximum amount the task is willing to pay). If the price is less than the task value then the task is allocated to the minimum price server and the task price set to the agreed price. Otherwise the task is removed from the set of tasks to be allocated. In the process of allocating a task to a server could result in some other tasks being deallocated from teh server. These deallocated tasks are then added back to the set of tasks to be allocated to be selected again.

Algorithm 3 Decentralised Iterative Auction

Require: I is the set of servers
Require: J is the set of unallocated tasks, that is initial the set of all tasks
Require: $P(i,k)$ is the 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: $\leftarrow_R$ will randomly select an element from a set
  while $|J| > 0$ **do**
    $j \leftarrow_R J$
    $p, i \leftarrow \min_{i \in I} P(i,j)$
    if $p \leq v_j$ **then**
      $p_j \leftarrow p$
      Allocate task j to server i
      Add tasks deallocated from server i back to J
    end if
    $J \leftarrow J \setminus \{j\}$
  end while

4.3.1 Server revenue optimisation problem. To find the optimal revenue for a server m given a new task n' and a set of currently allocated tasks N has a similar formulation to the optimisation problem in section 3.2. Except with an additional variable for the task price p_n for each task n.

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

s.t.

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

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

$$\sum_{\forall n \in N} (r_n^{'} + s_n^{'}) \cdot x_n + (r_{n'}^{'} + s_{n'}^{'}) \leq R_m, \tag{14}$$

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

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

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

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

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

The objective (Eq. (11)) is to maximize the price of all currently allocated tasks (not including the new task as the price is zero). The server resource capacity constraints (Eqs. (12), (13) and (14)) are similar to the constraints in the standard model set out in section 3.2 except with the assumption that the task $n^{'}$ is running. The deadline and non-negative resource speeds constraints (15, 16 , 17 and 18) are all the same equation as the standard formulation for all the tasks plus the new task. As this formulation only considers a single server, the task allocation constraint is not considered.

4.3.2 Decentralised Iterative Auction properties. For our proposed auction, we consider four important properties in auction theory.

- Budget balanced - True. Since the auction can run without an auctioneer, the auction can be run in a decentralised system with no ”middlemen” taking some money. This means that all revenue goes straight to the servers from the tasks.
- Individually Rational - True. 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 - False. The ability for tasks to profit from misreport task attribute is difficult to numerous reason: a task’s cannot determine the choices of other task for which server they will choose, the order task get priced in (this is random) and can’t lie about their value as this information isn’t revealed. Task’s can misreport it’s attributes to force other task’s to make decisions that would otherwise result in the misreporting task from being deallocated from a server. For example, if a task misreports some attributes it may result in it being cheaper for another task to select a different server. This would mean that the misreported task would not being deallocated that would otherwise would be. However for large scale systems, intentionally misreporting such attribute is extremely difficult to profit from. This is empirically shown in subsection 5.4.
- Economic efficiency - False. The allocation of task’s to server is completely random until server becomes full, because of this, the initial allocation and the random selection of task’s meaning that often task’s result in a local

maxima rather than the global maxima. As a result, the auction is not 100% economically efficient however the local optima are often close to the global maxima as shown in subsection 5.2.

While the auction is not incentive compatible and tasks do not pay the critical value, unlike the critical value auction, task's do pay the minimal amount for the task to be allocated. This is different from the critical value for two reasons: the critical value is not found through a deterministic process and the auction is not economically efficient resulting in the global maxima always being found possibly resulting in the task paying a different price each time the auction runs.

4.4 Attributes of the proposed algorithms

In this paper, we have presented three mechanisms to solve the optimisation problem proposed in section 3.2. Table 1 considers a range of important attributes of the proposed algorithm to allow easy comparison between the Greedy algorithm, Critical Value auction and Decentralised Iterative auction.

Attribute	Greedy Algorithm	Critical Value Auction	Decentralised Iterative Auction
Truthfulness		Yes	No
Optimality	No	No	No
Scalability	Yes	Yes	No
Task information requirements from users	All	All	All except the task value
Communication over heads	Low	Low	High
Decentralisation	No	No	Yes

Table 1. Attributes of the proposed algorithms: Greedy Algorithm, Critical Value Auction and the Decentralised Iterative auction

5 EMPIRICAL RESULTS

To evaluate the algorithms presented in Section 4, this section analyses the performance of both the greedy and auctions algorithms (subsections 5.1 and 5.2). We investigate the effect of server heuristics on the Decentralised Iterative Auction (subsection 5.3) and the possibility of misreporting task attributes within the Decentralised Iterative Auction (subsection 5.4). Finally we analyse the effectiveness of elastic resource allocation given server resource capacity ratios (subsection 5.5) and the comparison between online and batched resource allocation methods (subsection 5.6).

Within Edge Cloud Computing, we have found no de facto standard for testing resource allocation algorithms and those used in related work do not consider a deadline applicable for our work. Therefore we have generated synthetic models for our servers and tasks. This is done using a handcraft Gaussian distribution for each attribute that can be found in Appendix B.

In order to compare the results of the different algorithms, a branch and bound solution was implemented to solve the optimisation problem, from subsection 3.2. However due to the difficulty of finding the optimal solution, this algorithm was only used in smaller settings were the optimal could be found within a reasonable amount of time.
The settings chosen were 3 servers and 15 tasks, 6 servers and 30 tasks and 8 servers and 40 tasks where $\tilde{7}0\%$ of tasks are allocated.
To compare to previous work that utilise a fixed resource requirement, each task's resource speeds were determined by

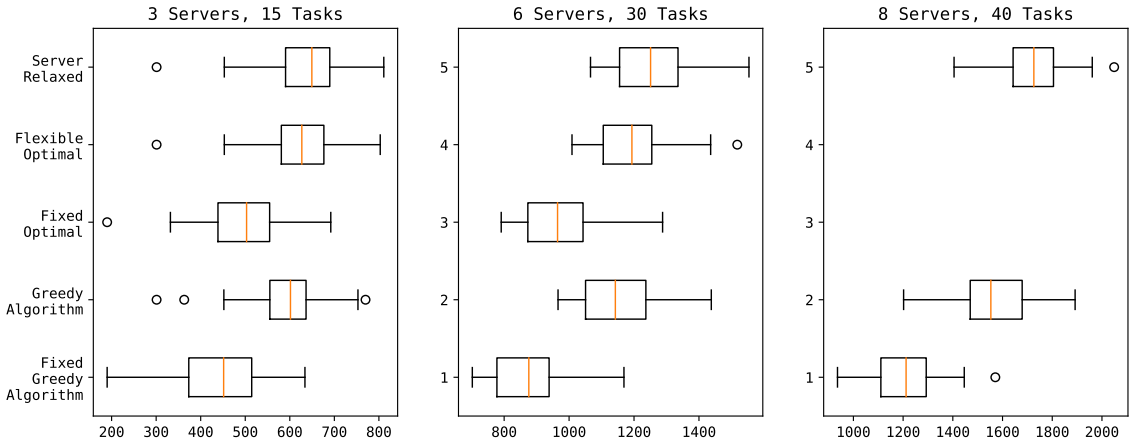

Fig. 5. Social welfare of the greedy algorithm, optimal flexible solution, optimal server relaxed solution, optimal fixed solution

finding the minimum sum total of resource speeds according to the task deadline. This aims to give each task a balanced amount of each resource making comparisons as fair as possible.

5.1 Evaluation of the Greedy Algorithm

In order to compare the greedy algorithm, the branch and bound solution above can be used to find the optimal solution for both the flexible and fixed resource allocation methods. However as explained above, to use the time required we are not able to use the solution for the larger problem setting (8 servers and 40 tasks). Therefore by modifying the greedy algorithm to use a fixed resource allocation we are able to find an approximate solution for the fixed resource allocation case in the large setting.

Along with this, a relaxed solution was implemented with the branch and bound algorithm containing just a single server that combined all of the server capacities into one. The optimisation problem for it can be found in Appendix C. This was used as the upper bound for the social welfare as the server is able to complete at least as much as all of the servers individual as well as use the remaining resources of the servers to possible compute more tasks.

For the greedy algorithm, a wide range of possible functions were tested with the best found being - value density function:

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

server selection:

$$\text{argmin}_{\forall i \in I} S_i^{'} \cdot W_i^{'} \cdot R_i^{'}$$

for task j and servers I and the resource allocation:

$$\text{argmin}_{s_j^{'}, w_j^{'}, r_j^{'}} \left(\frac{w_j^{'}}{W_i^{'}}\right)^3 + \left(\frac{s_j^{'} + r_j^{'}}{R_i^{'}}\right)^3$$

is task j and server i.

5.2 Evaluation of the Auction mechanisms

VCG, as explained in section 4, is the traditional method of dealing with self-interested users due to being incentive compatible. However, this requires solving the optimal solution for each individual task making such a mechanism infeasible except for small setting. Because of this, VCG is used for a comparison with the Critical Value Auction and Decentralised Iterative Auction in the smaller settings.

For the Critical Value Auction, the same function were used for the analyse of the greedy algorithm, in subsection 5.1. For the Decentralised Iterative Auction, the server heuristics are a price change of 3 and an initial price of 25 for all servers. The reason for these heuristics over others is explained in subsection 5.3.

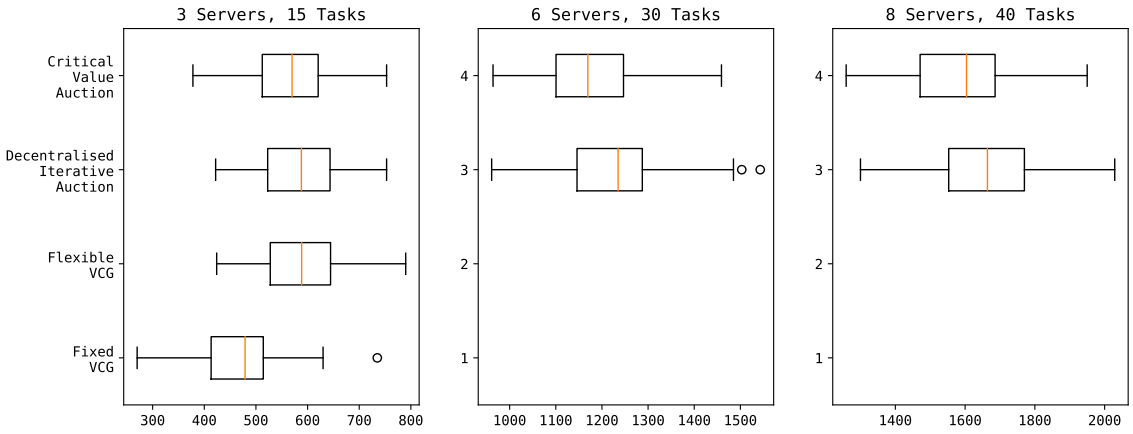

Fig. 6. Comparison of the social welfare for the auction mechanisms

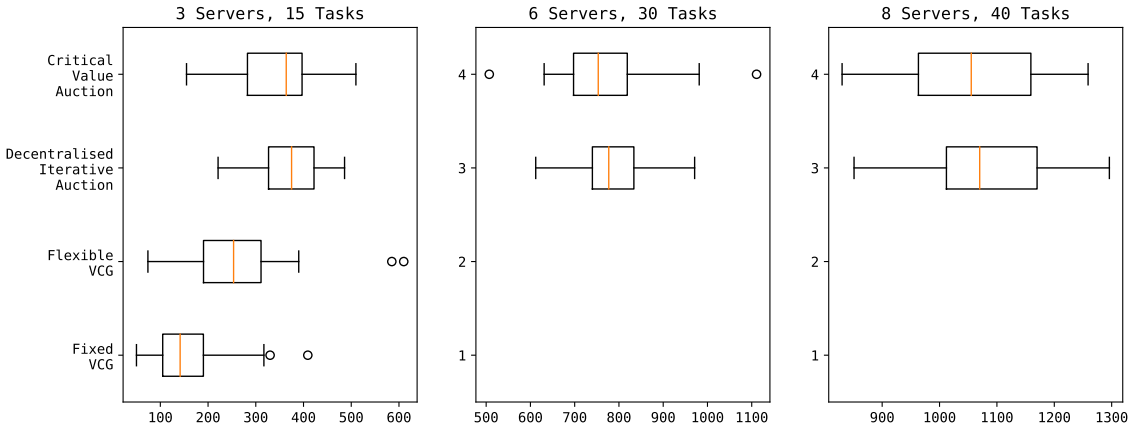

Fig. 7. Comparison of the social welfare for the auction mechanisms

5.3 Effectiveness of Decentralised Iterative Auction Heuristics

As explained in subsection 4.3, the Decentralised Iterative Auction calculates the price of task as the difference in revenue between the task being allocated (with a price of zero) and not being allocated. To increase a server revenue, a server will increase the price by a value referred to as the price change variable.

This is important as if a task has a value of 10 and the difference in revenue is 7 but the price change is 4 then the resulting price is 11, above the task's value prevent it from being allocated. As a result, the task won't be allocate to the server however if the price change was 3 then the task could have been allocated resulting in the server's total revenue benefiting both the server and the task in question.

A second heuristic, the initial price variable, is used to speed up the auction and reduce the number of rounds by aiming to set the price close to the task's actual value. This in turn reduces the number of rounds for the task price to converge on the task value. However, a similar problem as the price change variable exists such that if the initial price is greater than a tasks actual value then the task will never be allocated to a server.

Therefore the selection of price change and initial price can have a significant impact on the social welfare, revenue and number of rounds the auction takes. As the mean value of tasks is 50 (see Appendix B), the initial prices were set at 20, 25, 30, 35 and 40 while the price change variables were set at 1, 3, 5, 7, 10. These values were chosen to allow for understanding the auction where value at both ends of the spectrum are used for both variables, in order to see the effect on social welfare, revenue and number of rounds taken. For this case, we used 30 tasks and 6 servers.

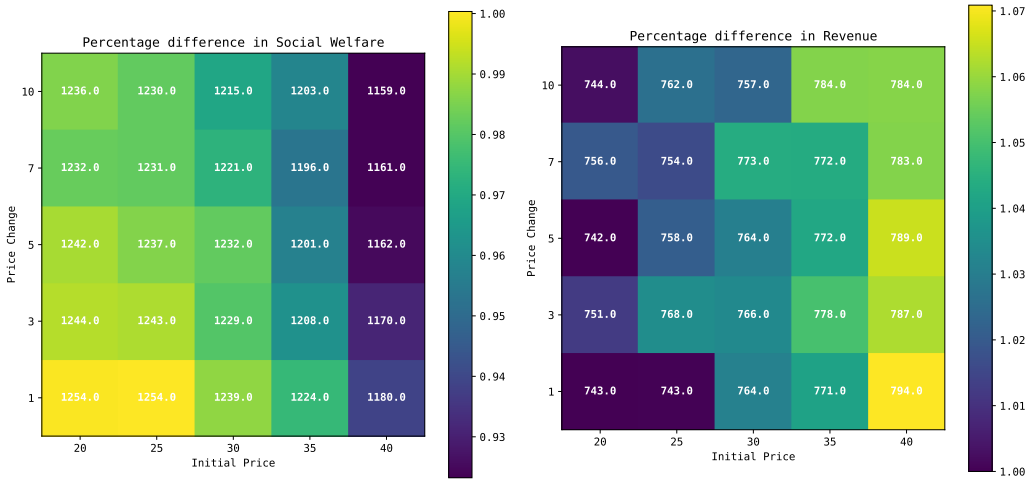

Fig. 8. Grid search of difference server price change and task initial cost effect on social welfare (left) and revenue (right)

Figure 8 maps a grid search of different price change and initial price values plotting the social welfare on the left and the total revenue on the right. The value are normalised with the result of the solution with a price change of 1 and initial price of 20. For the social welfare the percentage different between the best case (price change of 1 and initial price of 20) to the worst results (price change of 10 and initial price of 40) is only -7% meaning that the auction is resilient to a range of price change and initial price values.

While interesting when measuring the revenue, the lowest revenue occurs for the case of price change of 1 and initial price of 20. This is understandable as the system will arrive at a revenue local optima and for this case a majority of

tasks will pay their critical value however this also results in servers not gaining by forcing tasks by paying more than they could have. However for the case with an initial price of 40, this forces tasks off the bat to pay a price of 40, in most cases more than their critical value.

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 the time taken is proportional to the number of rounds run. Figure 9 shows the number of rounds required on average for each heuristic. For the case where the auction used the minimum heuristic values (price change of 1 and initial price of 20) that achieves the highest social welfare comes at the close of requiring on average 400 rounds. This means that on average tasks require XX number of individual rounds to converge on a price. In comparison when the price change is set to 10 and the initial price is kept at 20, the average number of rounds required is 5x less at 80.

This shows that the Decentralised Iterative Auction is an effective auction but has tradeoffs for server who must choose whether to maximise social welfare, revenue or rounds as each requires a different selection of heuristics.

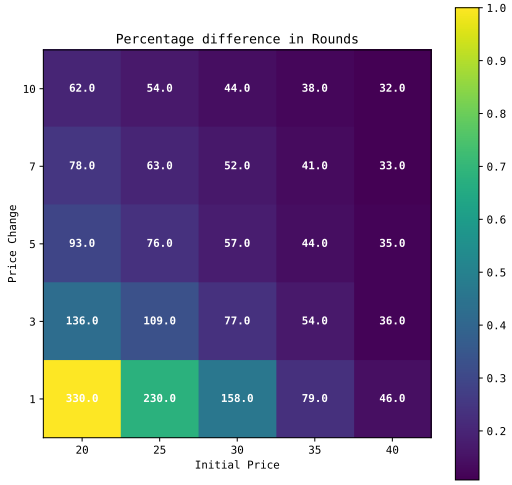

Fig. 9. Grid search of difference server price change and task initial cost effect on round count and number of task rounds

5.4 The Possibility of Misreporting Task Attributes in Decentralised Iterative Auction

In Subsection 4.3, the Decentralised Iterative Auction was shown to be not incentive compatible in specials cases however it is doubted that the ability for users to cheat more generally was possible. Therefore, this problem was addressed by applying a grid search of task attributes and to randomly modify individual tasks to investigate if tasks can pay less compared to if it didn't misreport.

For the random modification search, this was done by taking a random task and mutating each of the task's attributes.

5.5 The Effective of Elastic Resources given Server Resource Capacity Ratio

The major advantage of using elastic resource allocation is to the control that it gives to servers. This becomes increasing important if task requirements are unbalanced compared to server resource capacity. To measure the impact of elastic resource allocation compared to fixed resource allocation, server computation and bandwidth capacities are redistributed

to different ratios in order to compare. The reason only the computation and bandwidth capacities are modified is due to elasticity not affect the server's storage just its bandwidth and computation (eq 4 for loading and sending results and eq 3 for computation results).
Server resources are redistributed using a percentage of the server's total (computation + bandwidth) resources. This is shown as a ratio of server computation to bandwidth ratios in figures 10 and 11.

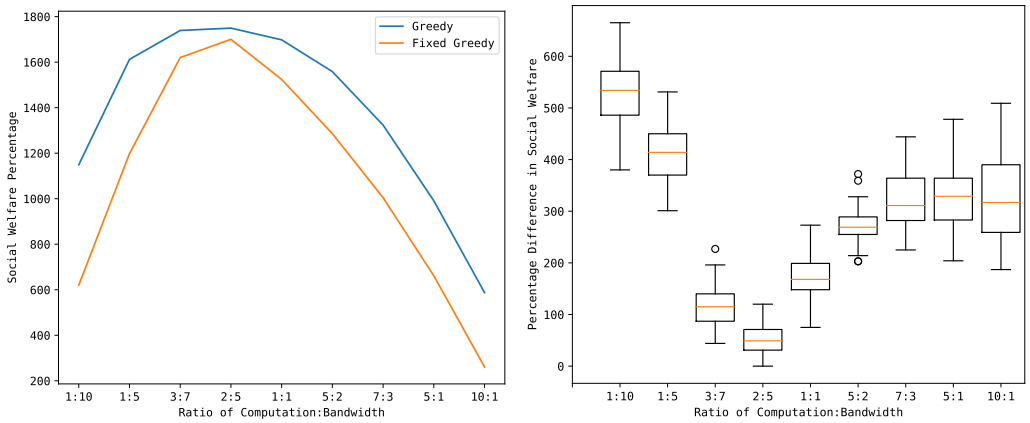

Fig. 10. The effect of server resource ratios on average social welfare percentage and the social welfare difference

Figure 10 is the combination of two graphs showing the same data. The left graph shows the average social welfare achieved by the respective algorithms with different server resource ratios. As can be seen on the figures at the ratio extremes, there is a significant difference in results between the flexible and fixed resource allocation. This can be seen more clear by the right graph, that plots the difference in social welfare per model with the error bars.
This observation is important for Edge Computing as nodes may have a large discrepancy in server resource capacity due several reasons, i.e. a persistence task running over a long period of time, internet connection issues limiting overall bandwidth, etc. For these cases, the elasticity of tasks has a significant boost to the number of tasks and in social welfare in comparison to fixed resource allocation.

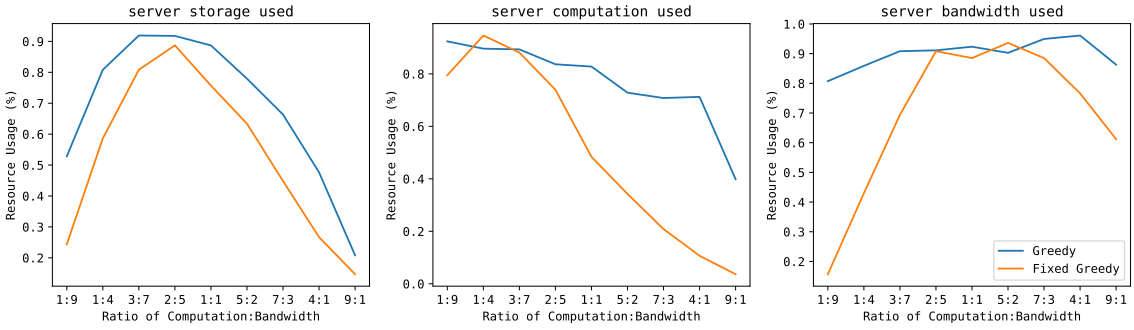

Fig. 11. Server resource usage

The effect of the redistribution of resources can be seen through the usage of resources by the servers per each ratio in figure 11. For cases where there is a large capacity of a resource available, for elastic resource allocation these resources are able to be used to maximise the number of tasks that can be run. This is particularly true of bandwidth as for a majority of the ratios, it is limiting factor for allocating tasks.

5.6 Comparison between Online and Batched Resource Allocation

The purpose of this work has to be explore to use of elastic resource allocation in a static one-shot environment meaning that all tasks arrive at the start, that are then allocated. However a problem exists with this formulation that in reality, task do arrive over time. Section 3 proposed tasks are run in batches overtime allowing no modifications to the algorithms proposed. This means that that as tasks arrive they are added to a queue, such that when a new batch occurs, all of the tasks in the queue can be processed using the proposed algorithms with no modifications.
Therefore choosing the right branch length is important as the more competition means better selection of tasks increasing social welfare while the longer the branch length reduces the length of the task processing time. For the Decentralised Iterative Auction, it has an advantage practically of being run in a batch as the auction doesn't need to wait for all of the tasks to arrive compared to the Critical Value Auction. Therefore the auction can run during the batch time frame with the task allocation at the end of the batch being the allocation.

We choose the batch lengths: 1, 2, 3, 4 because the mean length of a task is 10, any longer than this would result in tasks on average tasks having to be computed in half their possible time. Plus, as can be seen in figure 12, the total social welfare decreases as the batch lengths increase investigating longer batch lengths unwarranted.

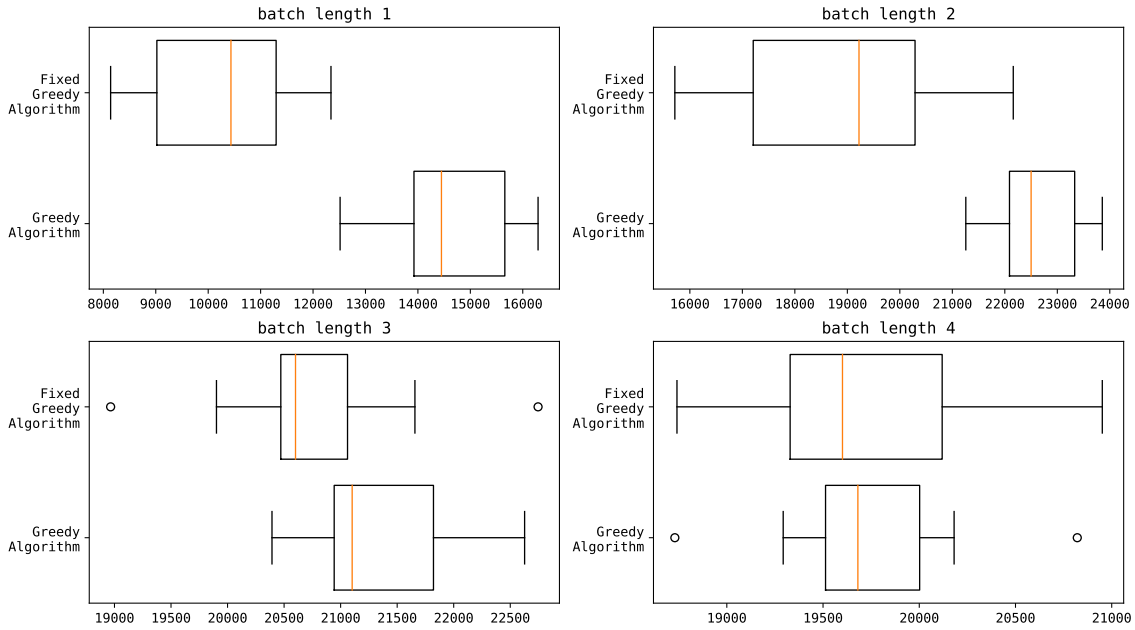

Fig. 12. Online batch lengths

6 CONCLUSION AND FUTURE WORK

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. 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, while this research considers online based mechanisms (subsection 5.6), this was not the primary environment these mechanisms were intended to occur within. Therefore additional research will focus primarily on this online scenario.

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* (Philadelphia, Pennsylvania, USA) *(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] X. Chen, W. Li, S. Lu, Z. Zhou, and X. Fu. 2018. Efficient Resource Allocation for On-Demand Mobile-Edge Cloud Computing. *IEEE Transactions on Vehicular Technology* 67, 9 (2018), 8769–8780.

[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] Tobias Dantzig. 1930. *Number: The Language of Science.*

[8] 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

[9] Mostafa Ghobaei-Arani, Alireza Souri, and Ali A Rahmanian. 2019. Resource management approaches in fog computing: A comprehensive review. *Journal of Grid Computing* (2019), 1–42.

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

[11] 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)*.

[12] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. 2013. Elasticity in Cloud Computing: What It Is, and What It Is Not. In *10th International Conference on Autonomic Computing (ICAC 13)*. USENIX Association, San Jose, CA, 23–27. https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst

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

[14] 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

[15] 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).

[16] G. B. Mathews. 1896. On the Partition of Numbers. *Proceedings of the London Mathematical Society* s1-28, 1 (11 1896), 486–490. https://doi.org/10.1112/plms/s1-28.1.486 arXiv:https://academic.oup.com/plms/article-pdf/s1-28/1/486/4323192/s1-28-1-486.pdf

[17] M. Mukherjee, S. Kumar, Q. Zhang, R. Matam, C. X. Mavromoustakis, Y. Lv, and G. Mastorakis. 2019. Task Data Offloading and Resource Allocation in Fog Computing With Multi-Task Delay Guarantee. *IEEE Access* 7 (2019), 152911–152918.

[18] 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

[19] D. T. Nguyen, L. B. Le, and V. Bhargava. 2018. Price-based Resource Allocation for Edge Computing: A Market Equilibrium Approach. *IEEE Transactions on Cloud Computing* (2018), 1–1.

[20] 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

[21] 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

[22] 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

[23] 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

[24] Sartaj Sahni. 1975. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM (JACM)* 22, 1 (1975), 115–124.

[25] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab. 2015. Edge computing enabling the Internet of Things. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 603–608. https://doi.org/10.1109/WF-IoT.2015.7389122

[26] 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)*.

[27] 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

[28] Argyrios G Tasiopoulos, Onur Ascigil, Ioannis Psaras, and George Pavlou. 2018. Edge-MAP: Auction markets for edge resource provisioning. In *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 14–22.

[29] Paolo Toth. 1980. Dynamic programming algorithms for the zero-one knapsack problem. *Computing* 25, 1 (1980), 29–45.

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

APPENDICES

Appendix A: Example problem case task and server attributes

Within subsection 3.4 and 4.1, this example problem case is used to demonstrate the effectiveness of elastic resource allocation compared to fixed resource allocation.

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

Table 2. 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 3. Table of task attributes. The columns ($s_j^{'}$, $w_j^{'}$, $r_j^{'}$) are fixed speeds which are not considered by the flexible resource allocation.

Appendix B: Synthetic Model

For the evaluation of the work in section 5, we used the following models. The left model for the majority of the analyse with static one-shot cases and the right model for the analysing the online and batch resource allocation methods.

Attribute Name	Mean	Standard Deviation
v_j	50	20
s_j	100	15
w_j	100	15
r_j	50	10
d_j	10	2
S_i	450	50
W_i	70	25
R_i	290	45

Table 4. Table of task and server attribute mean and standard deviations (used in section 5)

Attribute Name	Mean	Standard Deviation
v_j	50	20
s_j	100	15
w_j	100	15
r_j	50	10
d_j	10	2
S_i	400	40
W_i	40	10
R_i	150	20

Table 5. Table of task and server attribute mean and standard deviations for the online work (used in subsection 5.6)

Appendix C: Server Relaxed Optimisation Problem

$$\max \sum_{\forall j \in J} v_j x_j \tag{20}$$

s.t.

$$\sum_{\forall j \in J} s_j x_j \leq S_i, \tag{21}$$

$$\sum_{\forall j \in J} w_j^{'} x_j \leq W_i, \tag{22}$$

$$\sum_{\forall j \in J} (r_j^{'} + s_j^{'}) \cdot x_j \leq R_i \tag{23}$$

$$\frac{s_j}{s_j^{'}} + \frac{w_j}{w_j^{'}} + \frac{r_j}{r_j^{'}} \leq d_j, \qquad \forall j \in J \tag{24}$$

$$0 < s_j^{'}, \qquad \forall j \in J \tag{25}$$

$$0 < w_j^{'}, \qquad \forall j \in J \tag{26}$$

$$0 < r_j^{'}, \qquad \forall j \in J \tag{27}$$

$$x_j \in \{0, 1\}, \qquad \forall j \in J \tag{28}$$