

# 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 processed at the edge of the network using limited computational resources in comparison to larger remote data centres. Because of this, resource allocation and management is significantly more important. Existing resource allocation 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 resulting in resource bottlenecks. To address this, we propose a novel approach that takes advantage of the elastic nature of resources as the time taken for an operation to occur is proportional to the resource allocated. This, however, makes previous research incompatible with this problem. 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 mechanism 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), 19 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 high security and privacy. To deal with such domains, *fog/edge computing* [6] 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 clouds*.

As the Internet-of-things grows, fog/edge cloud computing is a key enabling technology [6, 24], in particular for applications in smart cities [25], 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.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2020 Association for Computing Machinery.

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 to minimise vehicle waiting times or to analyse video feeds from CCTV cameras [26]. 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 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 overrequest particular resources.

Previous work [12] has considered the ability of cloud servers to flexibly respond to usage by expanding resource capacity when needed. Due to the often ad hoc nature of fog cloud networks, such an ability is generally not feasible, therefore in this work we utilise the ability for tasks to have flexibility in how its resources are allocated due to the linear relationships of many task attributes.

An example is the time taken for a program to be download by the server is generally proportional to the bandwidth allocated. This 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 leaves for future work, the ability for task's to be considered that scale non-linearly.

Using this capability to flexibly allocate resources is additionally important in the case of edge computing due to the limited resources capacities that servers have in comparison to the large data centres used in standard cloud computing. Therefore, using task resources flexibility, we propose a new resources allocation optimisation problem that enables servers to have greater flexibility over how its resources are allocated. However, this new optimisation problem is incompatible with previous research in this area, thus we propose three mechanisms that utilise this flexibility.

In the following section, an overview of related work is provided. Section 3 presents the problem formulation, an optimisation problem, and an example case to show the effectiveness of such a system. Using this formulation, Section 4 presents three different algorithms: a modular greedy algorithm, a centralised incentive-compatible auction, and a novel decentralised iterative auction. Using these algorithms, Section 5 presents an extensive analysis of our mechanisms compared to the optimal task, flexible solution and a strict resource requirement solution. .

## 2 RELATED WORK

Within Fog Computing, a range of approaches have been proposed for 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 are explained.

Auctions are a popular method for dealing with self-interested users to limit their ability to gain utility due to misreporting information. This is also a helpful method of determining the value when allocating multiple limited resources. Edge-MAP [27] provides a client-to-cloud model for tasks with short deadlines (10-50ms). Through using a Vickrey-English-Dutch (VED) auction, the system arrives at the unique minimum competitive equilibrium price. Using this, the system is highly scalable and adaptable to dynamically changing network topologies with cloudlet nodes at the edge of the network.

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

Within auctions in cloud computing there is a large corpus of research, [14] provides a systematic study of double auction mechanisms that are suitable for a range of distributed systems like Grid computing, Cloud computing, Inter-Cloud systems. 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.

An advantage of fog nodes is that due to the proximity, nodes can find other nodes in its vicinity to assist in a task. Using this ability, [17] has propose a resource allocation mechanism to allow fog nodes to offload multiple tasks with delay guarantees. This is helpful due to the limited resources of nodes, thus the system allows for both other fog nodes or remote cloud centers for further computational resources. This causes an issue to decide 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. [4] propose a resource-efficient computation offloading mechanism for users, where communication and computation are joined by the node operation when allocating resources. 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 [8], 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. 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 allows the algorithm to run in polynomial time complexity.

Alternative work by [3] to efficiently distribute data centers and connections aims to maximise social welfare instead of the number of tasks completed. 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 as a VM of different specifications. 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.

Our problem is related to multidimensional knapsack problems where there is a large body of works [7, 16]. In particular [20], considers the flexibility of item weights with a linear constraint that must be fulfilled for the item to be allocated. The paper provides a pseudo-polynomial time complexity algorithm for solving this problem to maximize the values in the knapsack. Our optimisation problem case is similar to their, but differs due to constraints with our using non-linear not linear constraints.

### 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  $I = \{1, 2, \dots, |I|\}$ , 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  $R_i$ .

The system also contains a set  $J = \{1, 2, \dots, |J|\}$ , 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 to compute the task. In order to run a task requires the server to load the appropriate code/data on 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.

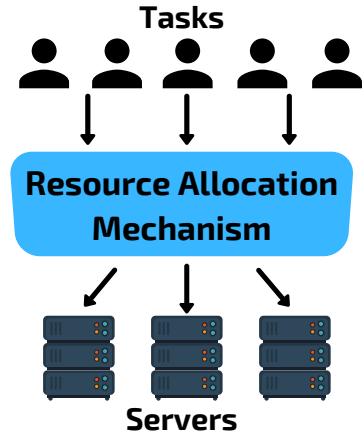


Fig. 1. System Model

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  $i$ .

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

s.t.

$$\sum_{\forall j \in J} s_j x_{i,j} \leq S_i, \quad \forall i \in I \quad (2)$$

$$\sum_{\forall j \in J} w'_j x_{i,j} \leq W_i, \quad \forall i \in I \quad (3)$$

$$\sum_{\forall j \in J} (r'_j + s'_j) \cdot x_{i,j} \leq R_i, \quad \forall i \in I \quad (4)$$

$$\frac{s_j}{s'_j} + \frac{w_j}{w'_j} + \frac{r_j}{r'_j} \leq d_j, \quad \forall j \in J \quad (5)$$

$$0 < s'_j, \quad \forall j \in J \quad (6)$$

$$0 < w'_j, \quad \forall j \in J \quad (7)$$

$$0 < r'_j, \quad \forall j \in J \quad (8)$$

$$\sum_{\forall i \in I} x_{i,j} \leq 1, \quad \forall j \in J \quad (9)$$

$$x_{i,j} \in \{0, 1\}, \quad \forall i \in I, \forall j \in J \quad (10)$$

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, with longer running tasks reappearing on consecutive time intervals. In subsection 5.6 evaluates the effectiveness of such a batching mechanism compared to an online mechanism. We leave a detailed study of online mechanisms where the resource allocated can dynamically change at each time to future work.

### 3.3 Time Complexity

As the optimisation problem as described in Subsection 3.2 is an extension of the Knapsack problem which is well-studied problem in computer science that known to be NP-Hard. Therefore, the time complexity of the problem is NP-Hard as well due to the problem being an extension of Knapsack problem.

**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.  $\square$

### 3.4 Example Problem Case

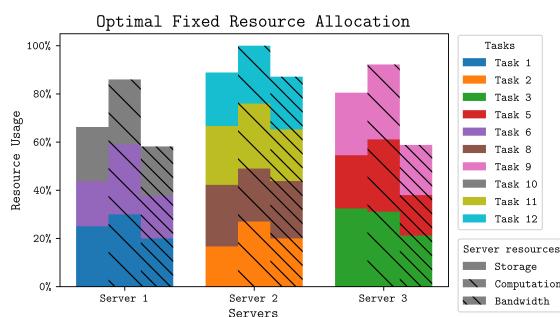


Fig. 2. Optimal solution with fixed resource speeds

have fixed resource speeds (which were set by minimising the total amount of resources to be completed within the 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.

Manuscript submitted to ACM

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 with the flexible solution able to achieve 18% better social welfare compared to the fixed resource allocation solution. The exact settings can be found in Appendix A with table 3 for the task attributes and table 2 for the 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

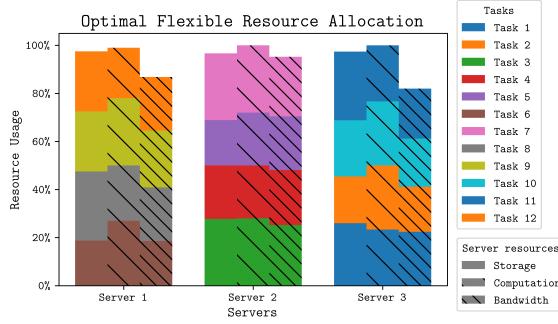


Fig. 3. Optimal solution with elastic resources speeds

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. 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 with each stage using an 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, 28] 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, an incentive-compatible auction using the centralised greedy algorithm (Section 4.2). While the second is 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 that we have extended to this problem specificity in subsection 3.2. Because of this elastic nature of task resources means that an additional stage is required to determine these speeds.

More specifically, the greedy algorithm has two stages; stage one sorts the list of tasks based on the value density of each task that is calculated based on 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.

Using the 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 resource.

---

**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 the loading, compute and sending speeds  
**Require:**  $\text{sort}(X, f)$  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  $f$

$$\begin{aligned} J' &\leftarrow \text{sort}(J, v) \\ \text{for all } j \in J' \text{ do} \\ &\quad i \leftarrow s(j, I) \\ &\quad \text{if } i \neq \emptyset \text{ then} \\ &\quad\quad s'_j, w'_j, r'_j \leftarrow \gamma(j, i) \\ &\quad\quad x_{i,j} \leftarrow 1 \\ &\quad \text{end if} \\ \text{end for} \end{aligned}$$


---

**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 mechanism is  $\frac{1}{n}$  of the optimal social welfare.*

**PROOF.** Due to a task not considering other task's resource requirements 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

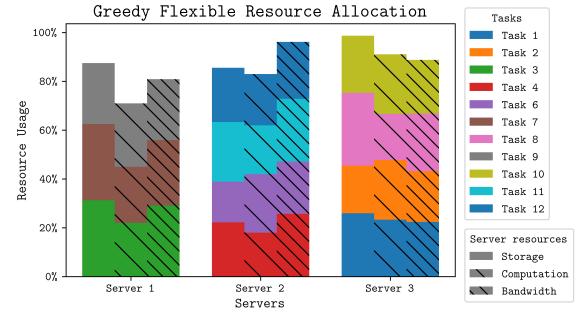


Fig. 4. Example Greedy allocation using the model from table 3 and 2

algorithm can be guaranteed to achieve at least  $\frac{1}{n}$  of the optimal social welfare through using a value density function ( $v(j) = j_v$ ). 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.  $\square$

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

**THEOREM 4.2.** *Time complexity of the greedy mechanism 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 mechanism 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  $O(|J| |I| + |J| \log(|J|)) = O(|J| |I|)$ .  $\square$

## 4.2 Critical Value Auction

Due to the problem case of being non-cooperative, if the greedy mechanism 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 a user to honestly reveal a task's attribute.

The auction is implemented using the greedy mechanism from section 4.1 by finding an initial allocation using every task's reported value. Then for each task that is allocated, the task price is equal to the critical value is found by finding the minimum value of the task such that it is still allocated.

To find the minimum value is a two-step process of removing the task from the list of tasks then running the greedy mechanism however after each task is allocated. Then a check is done if the critical task could be allocated to any server. This is a constant time complexity operation by assuming that the server would allocate all of its available resources for the deadline constant (eq 5). If the task can't be allocated to any server then the value density of last task allocated is equalled to the required value density of the critical task (this assumes that in the sorted list, the critical task would appear ahead thus a minor amount could be add thus to guarantee the critical task is above). Using the value density and the value density function, through finding the inverse of the value density function, with regards to the task value allows the task critical value to be calculated.

**4.2.1 Critical Value Auction Time Complexity.** The time complexity of the auction is  $O(|J| |J| |I|)$ , the greedy mechanism 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.** The auction uses the greedy mechanism, who has time complexity is  $O(|J| |I|)$ , (subsubsection 4.1.2) to find all task's that are allocated to a server. Using the list of all task's allocated, the critical value of each task must be found. This is done by repeating the server selection and resource allocation each all task (excluding the critical task) with time complexity,  $O(|J| |I|)$ , till the critical task can no longer be allocated to any server (a constant 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 possibly being found for every task.  $\square$

**4.2.2 Critical Value Auction Strategyproof.** In order that the auction is strategyproof, the value density function must be monotonic [22] 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 such that the resulting auction is strategyproof.

**THEOREM 4.4.** *The value density function  $\frac{v_j d_j}{\alpha(s_j, w_j, r_j)}$  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  $\alpha$  function is will increase as the resource requirements increase, the resulting value density will decrease.

So in any case, the overall value density will decrease if the owner doesn't accurately report a task's attribute, resulting in the task paying more.  $\square$

### 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, 28]. VCG auction calculates the price of an item by finding the difference in social welfare if the item exists and does not 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 required to be allocated with a price of zero. To cause the overall revenue and servers 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 section 4.3.1 plus a small value referred to as the price change variable. The reason for the price change variable is to increase the revenue of the server (otherwise the total revenue of the server doesn't increase by accepting the task) and is to be chosen by the server. Once all of the servers have responded, the task can compare the minimum server prices to its private value. If the price is less then the task will accept the server with the lowest price, otherwise the task must stop advertising as the price for the task to run on any server is greater than its reserve price preventing the task from ever being allocated again.

The algorithm 2 is a centralised version of the auction. It works through iteratively checking a currently unallocated task to find the price if the task 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 minus the new revenue with the task allocated plus a price change variable in order to increase the revenue of the server. The minimum price returned by  $P(i, k)$  is then compared to the task's maximum reserve price (that would be private in the equivalent decentralised algorithm) to confirm if the task is willing to pay at that price. If the task is willing then the task is allocated to the minimum price server and the task price set to

the agreed price. However in the process of allocating a task then the currently allocated tasks on the server could be unallocated so these tasks allocation's and price's are reset then appended to the set of unallocated tasks.

---

**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
     $j \leftarrow_R J$ 
     $p, i \leftarrow \text{argmin}_{i \in I} P(i, j)$ 
    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

```

---

**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  $n$ .

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 so there is no need to consider if it is running or not. 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.

$$\max \sum_{\forall n \in N} p_n x_n \quad (11)$$

s.t.

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

$$\sum_{\forall n \in N} w_n' x_n + w_{n'} \leq W_m, \quad (13)$$

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

$$\frac{s_n}{s_n'} + \frac{w_n}{w_n'} + \frac{r_n}{r_n'} \leq d_n, \quad \forall n \in N \cup \{n'\} \quad (15)$$

$$0 < s_n' < \infty, \quad \forall n \in N \cup \{n'\} \quad (16)$$

$$0 < w_n' < \infty, \quad \forall n \in N \cup \{n'\} \quad (17)$$

$$0 < r_n' < \infty, \quad \forall n \in N \cup \{n'\} \quad (18)$$

$$x_n \in \{0, 1\} \quad \forall n \in N \quad (19)$$

**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 resulting in no "middlemen" taking some money meaning 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. While a task's cannot determine the choices of other task for which server they will choose, the order task pricing as this is random or lie about the task 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 task misreports some attributes it may result in it being cheaper for another task to select a different server. This would mean that misreported task from not being deallocated. 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 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 that task's 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 due to the requirement that the value is found through a deterministic process, however as this auction randomly selects task's from the set of unallocated tasks to find a task it can't be the task's critical value.

#### 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 mechanism, 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 overheads	Low	Low	High
Decentralisation	No	No	Yes

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

## 5 EMPIRICAL RESULTS

To evaluate the algorithms presented in Section 4, this sections analysis the different algorithms, the possibility of misreporting task attributes, the effect of the server resource capacity and online task arrival.

To do evaluate our algorithms, synthetic models have been used to generate servers and tasks where each attribute was taken from a Gaussian distribution. The reason this was done was due to there not being a de facto standard to test cloud computing resource allocation algorithms and that those used in related work do not consider a deadline for a task.

### 5.1 Evaluation of the Greedy Algorithm

To compare the greedy algorithm to the optimal elastic allocation, a branch and bound was implemented to solve the optimisation 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  $\text{argmin}_{\forall i \in I} S'_i + W'_i + R'_i$  and the resource allocation was  $\min s'_j + w'_j + r'_j$  for job  $j$  and servers  $I$ .

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.

### 5.2 Evaluation of the Auction mechanisms

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

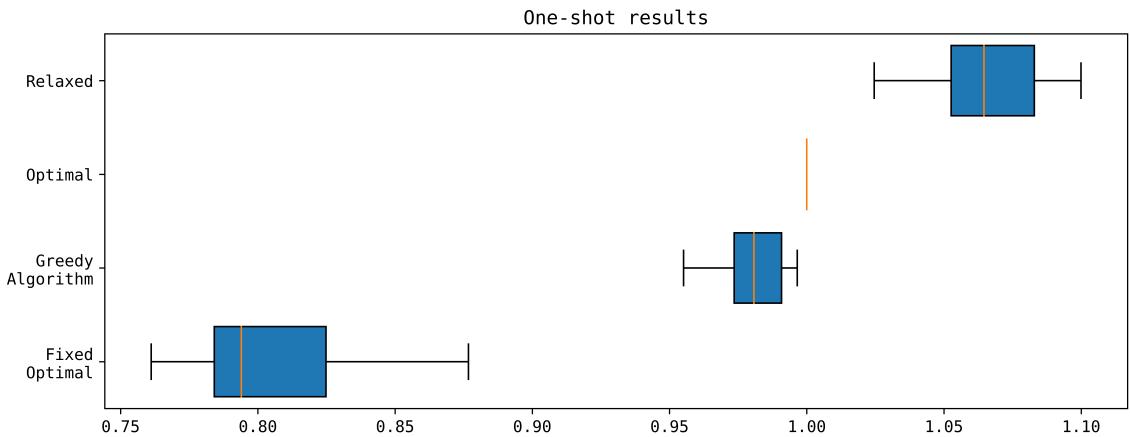


Fig. 5. Comparison of the social welfare for the greedy mechanism, optimal, relaxed problem, time limited branch and bound

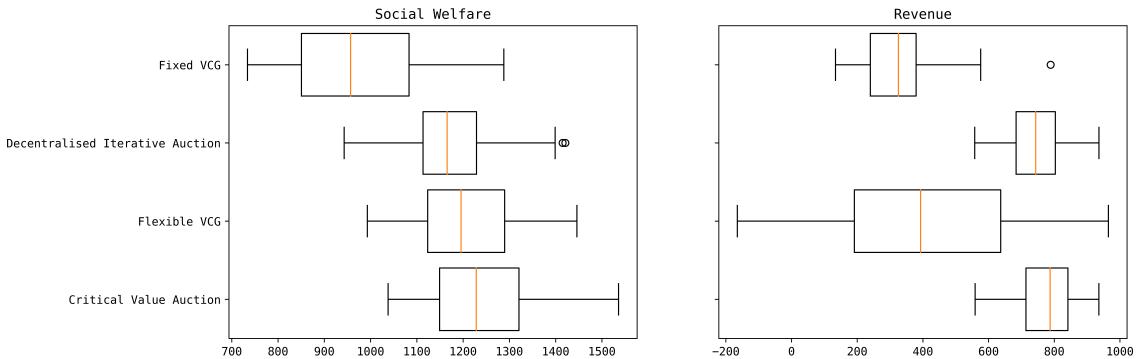


Fig. 6. Comparison of the social welfare for the auction mechanisms

### 5.3 Effectiveness of Decentralised Iterative Auction Heuristics

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 8, 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,

Manuscript submitted to ACM

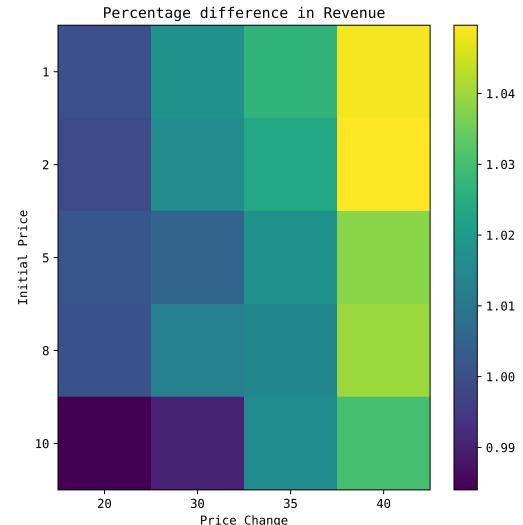


Fig. 8. Grid search for rounds

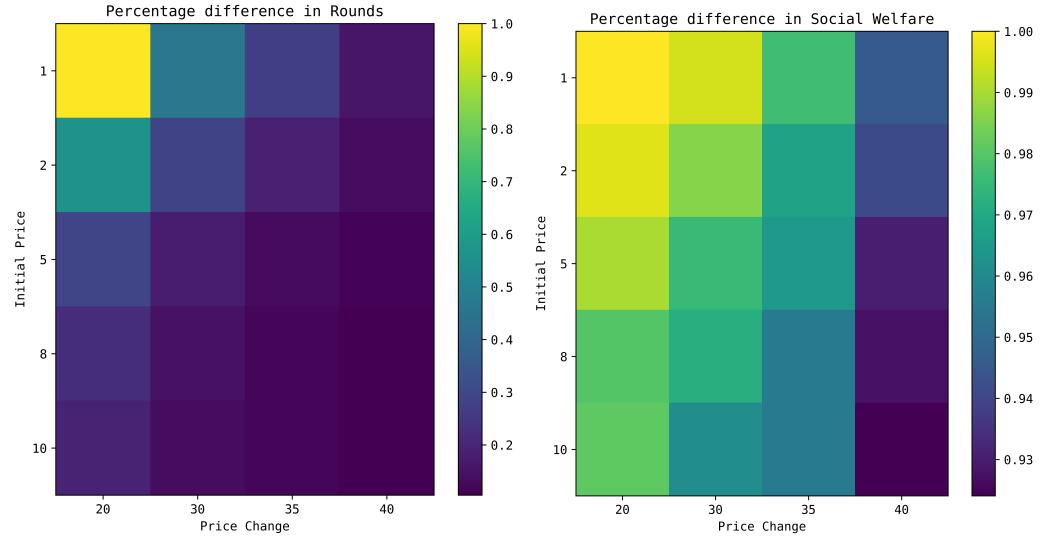


Fig. 7. Grid search of difference server price change and task initial cost

figure 7, shows that the difference in social welfare is only 2% from minimum to maximum heuristics.

#### 5.4 Possibility of Task Mutation in Decentralised Iterative Auction

#### 5.5 Effect of Server Resource Capacity Ratio

Due to the elasticity in the resources, an advantage of such a system is the ability for server to more efficiently distribute their resources particularly when certain server resources are scarce. To confirm this, a models were generated where for a range of ratios, a server's bandwidth and computation resources are redistributed to fit the ratio. At such a point, the greedy algorithm using the settings from subsection 5.1 were used, the optimal flexible solution and the optimal fixed solution.

This can be seen more clear by plotting the average server resource usage each of the ratios.

#### 5.6 Batching verse Online task allocation

\* Explain the reason for the evaluation \* Add figure \* Explanation for DIA advantage as can run over the batch not just at the end

## 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

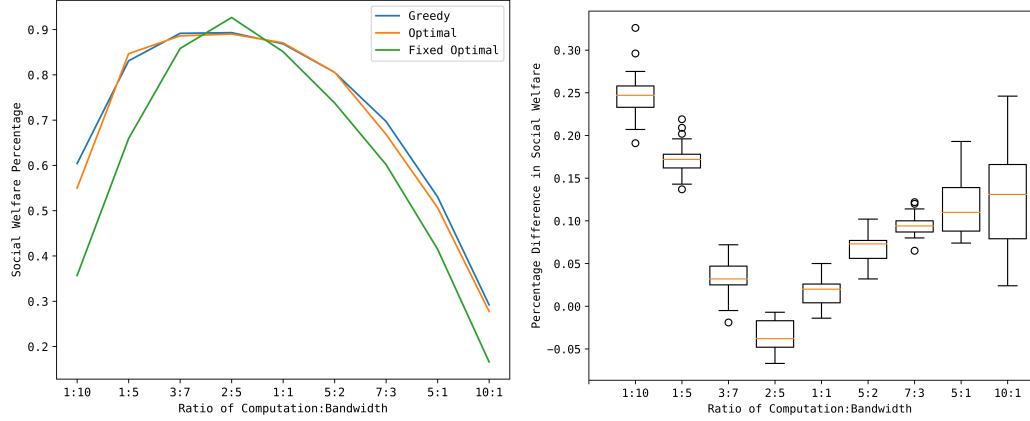


Fig. 9. Social welfare

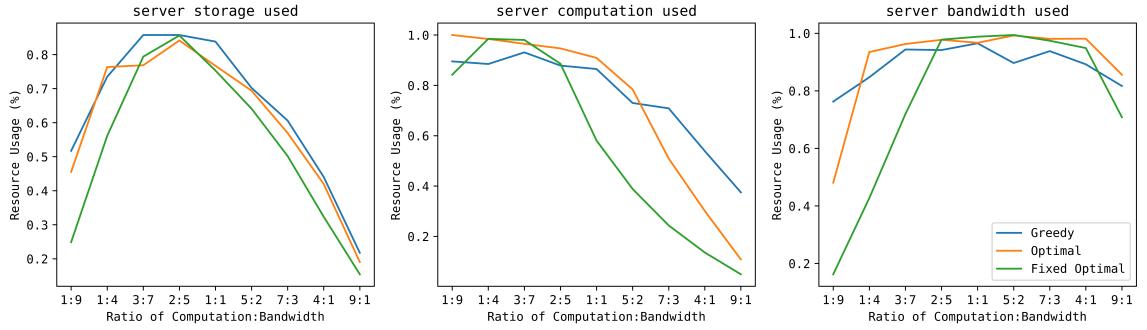


Fig. 10. Server resource usage

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, Mohammad B. Abdjabar, 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 Gerdin, 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.

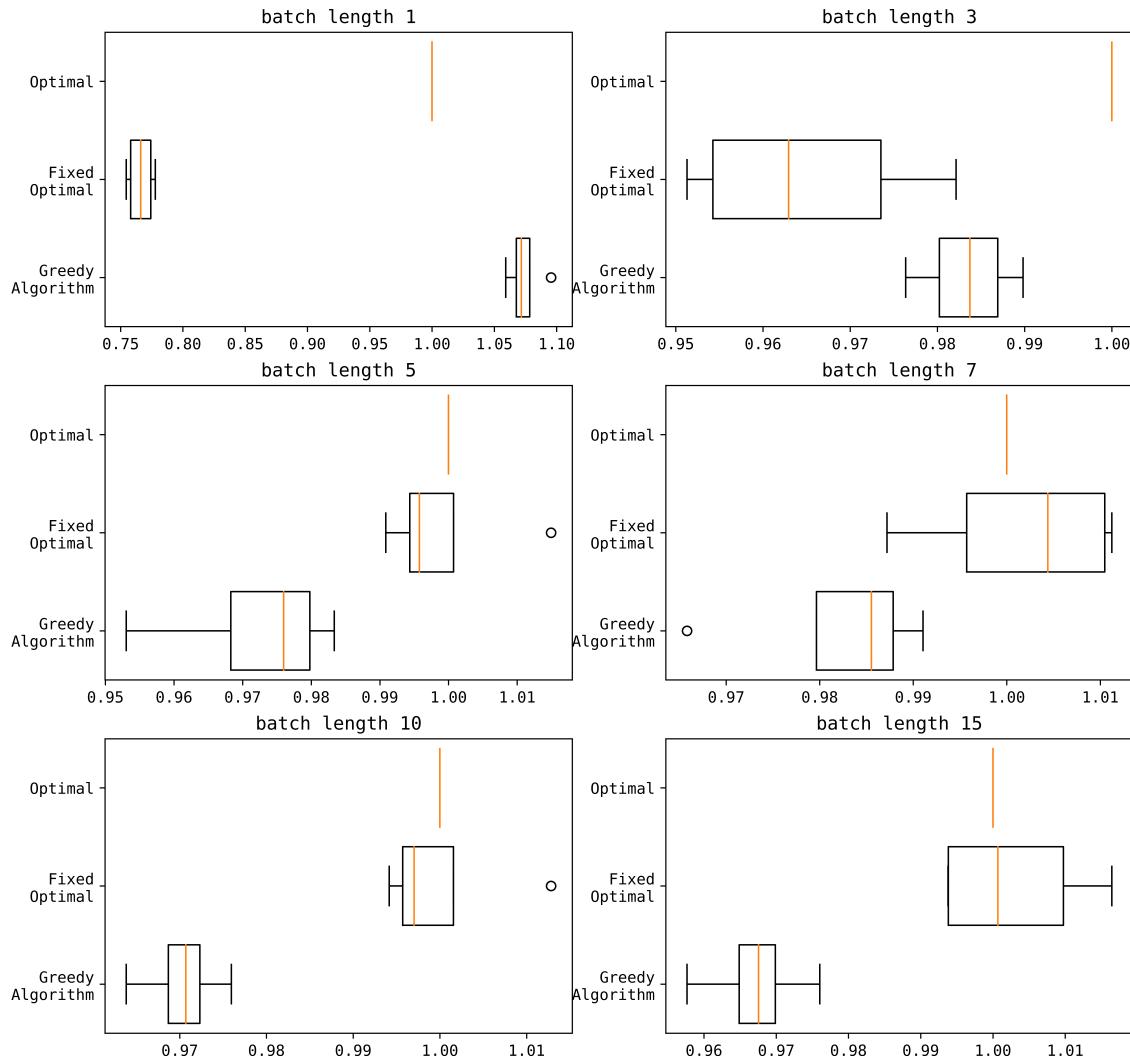


Fig. 11. Online batch lengths

- [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 Rahmani. 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/ijarcse.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] 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>
- [25] M. Sapienza, E. Guardo, M. Cavallo, G. La Torre, G. Leonbruno, 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)*.
- [26] 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>
- [27] 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.
- [28] William Vickrey. 1961. Counterspeculation, Auctions, and Competitive Sealed Tenders. *The Journal of Finance* 16, 1 (1961), 8–37. <http://www.jstor.org/stable/2977633>

## APPENDIX

### Example problem case task and server attributes

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.