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 resulting in resource bottlenecks. To address this, we propose a novel approach that takes advantage of the elastic nature of resource as the time taken for an operation to occur is proportional to the resource allocated. This, however, make previous research incompatible with this problem. Therefore, we describe this problem formally, show that it is NP-Hard, 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 but is not incentive compatible. In 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), 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the last few years, cloud computing [1] has become a popular solution for running data-intensive applications remotely. However, large scale data-centres may not a feasible for application domains that require low latency or high security and privacy. 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 clouds*.

As the Internet-of-things grows, fog/edge cloud computing is a key enabling technology in particular for applications in smart cities, disaster response scenarios, home automation systems, etc. In these applications, low-powered devices generate data or tasks that can't be processed locally but are impractical to use with standard cloud computing services.

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

More specifically, in smart cities, these devices could be used control smart traffic light systems which collect data from roadside sensors to optimise the traffic light sequence to minimise vehicle waiting times or to analyse videos feeds from CCTV cameras. In disaster response, 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 [5]. 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 [9], we are interested in allocating tasks to edge cloud servers to maximize the overall social welfare achieved (i.e., the sum of completed task values). This is particularly challenging, because users in edge clouds are typically self-interested and may behave strategically [2] or may prefer not to reveal private information about their values to a central allocation mechanism [13].

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 allocate 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.
This work utilises the ability for tasks to have flexibility in how its resources are allocated due to the linear relationships of many task attribute. 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 scalability. We leaves for future work, the ability for task's to be considered that scale non-linearly.

Using this ability to flexibility allocate resource is additionally important in the case of edge computing due to the limited resource capacities that servers have in comparison to large data-centre used in standard cloud computing. Therefore using task resource flexibility, we propose a new resource allocation optimisation problem that enables servers to have greater flexibility over how it's resource 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 algorithm: a modular greedy algorithm, a centralised incentive compatible auction and a novel decentralised iterative auction. Using these algorithms, Section 5 presents extension analysis such mechanisms compared to the optimal task flexible solution and a strict resource requirement solution. .

2 RELATED WORK

There is a considerable amount of research in the area of resource allocation and pricing in cloud computing, some of which use auction mechanisms to deal with competition [2, 3, 8, 15]. However, these approaches assume that users request a fixed amount of resources system resources and processing rates, with the cloud provider having no control over the speeds, only the servers that the task was allocated to. In our work, tasks' owners report deadlines and overall

data and computation requirements, allowing the edge cloud server to distribute its resources more efficiently based on each task's requirements.

Other closely related work on resource allocation in edge clouds [5] considers both the placement of code/data needed to run a specific task, as well as the scheduling of tasks to different edge clouds. The goal there is to maximize the expected rate of successfully accomplished tasks over time. Our work is different both in the setup and the objective function. Our objective is to maximize the value over all tasks. In terms of the setup, they assume that data/code can be shared and they do not consider the elasticity of resources.

Our problem is related to multidimensional knapsack problems. In particular [10], consider flexibility in the allocation, with linear constraints that are used for elastic weights. The paper provides a pseudo-polynomial time complexity algorithm for solving this problem to maximize the values in the knapsack. Our optimisation problem case is similar to their, but differ due to constraints with our using non-linear not linear constraints.

A majority of approaches taken for task pricing and resource allocation in Cloud Computing uses a fixed resource allocation mechanism, such that each user requests a fixed amount of resources for a task from a server. However this mechanism, as previously explained, provides no control for the server over the quantity of resource allocated to a task, only determining the task's price. As a result, a majority of approaches don't consider the server's management of resource allocation. Thus research has focused on designing efficient and incentive compatible auction mechanisms.

Work by [8] 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 server should act truthfully.

Some approaches have been taken to increase flexibility within Fog Cloud Computing [2] through efficient distribution of data centers and connections to maximise social welfare. 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. Similar research in [5], 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. An approximation algorithm achieved 90% of the optimal social welfare by converting the problem to a set function optimisation problem.

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 the effectiveness of our flexible resource allocation scheme compared to a 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, computation capacity with W_i, and the bandwidth capacity with R_i.

The system also contains a set $J = \{1, 2, \ldots, |J|\}$ 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.

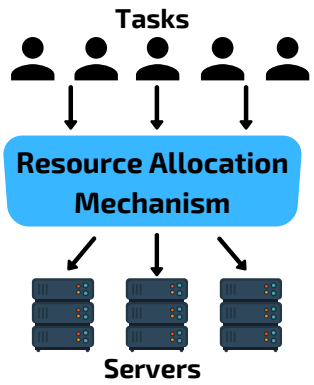


Fig. 1. System Model

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 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 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 fo 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 constraint 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 re-appearing on consecutive time intervals. In subsection 5.6 evaluates the effectiveness such a batching mechanism compared to an online mechanism. We leave a detailed study of online mechanisms where the resource allocate 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 satisfy the deadline constraint. This reduces the model to a 0–1 multidimensional knapsack problem [7], which is a generalization of a simple 0–1 knapsack problem. The latter is an NP-hard problem [7]. Given this, it follows that the 0–1 multidimensional knapsack problem is also NP-hard. Since 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

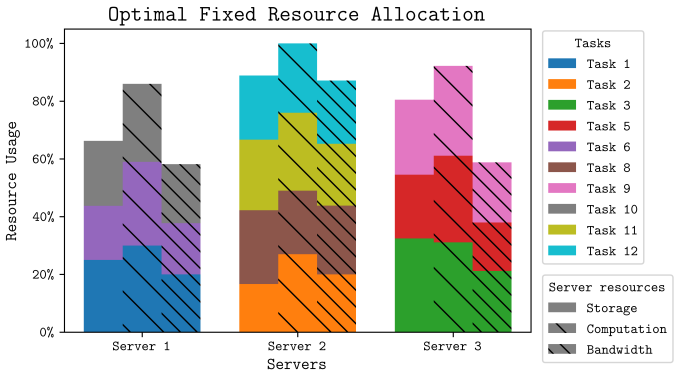


Fig. 2. Optimal solution with fixed resource speeds

Figure 2 shows the best possible allocation if tasks have fixed resource speeds (that 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.

In contrast to figure 2, Figure 3 depicts the optimal allocation if elastic resources are considered. Here, all of the resources are used by the servers whereas the fixed example 2 cant do this. In total, the elastic approach manages

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 represents each server as a group of three bars, each relating to the each server's resource types, with the percentage of resources used by a task being the size of the bar.

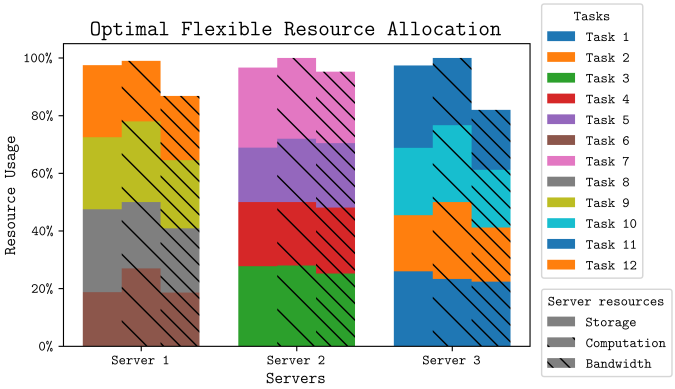


Fig. 3. Optimal solution with elastic resources speeds

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 is infeasible due to both the space and time complexity required.

Because of this issue of allocating both tasks to server and server resources to tasks, this work proposes an approximation algorithm where tasks are allocated to 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 [4, 6, 14] is used for such system 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 user, we propose two auction-based mechanisms, an incentive compatible auction using the centralised greedy algorithm (Section 4.2). While the second is a one of which is a novel decentralized iterative auction (Section 4.3) that do 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 calculate 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

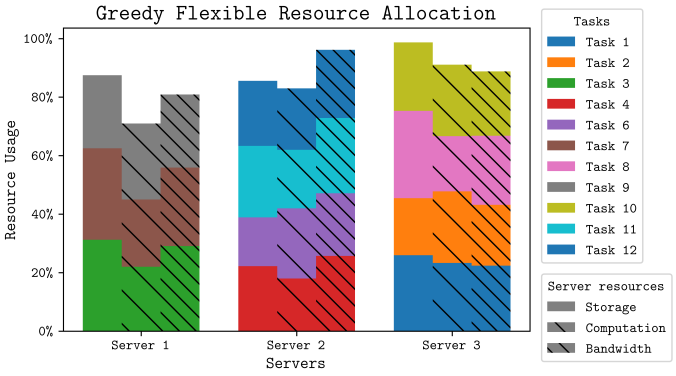


Fig. 4. Example Greedy allocation using the model from table 3 and 2

the tasks achieving XX% more social welfare
than the fixed solution. This is due to the al-
gorithm 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: 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
```

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 no 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 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, 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 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 the stage 1 of the mechanism is $O(|J|\log(|J|))$ due to sorting the tasks and stage 2 has complexity $O(|J|\,|I|)$ due to looping over all of the tasks and applying the server selection and resource allocation heuristics. Therefore the overall time complexity is $O(|J|\,|I| + |J|\log(|J|) = O(|J|\,|I|)$. □

4.2 Critical Value Auction

Due to the problem case being non-cooperative, if the greedy mechanism was used to allocate resources such that the value is the price paid. This 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 strategyproof (weakly-dominant incentive compatible) so users have no incentive to misreport task attributes.

Single-Parameter domain auctions are extensively studied in mechanism design [11] and are used where an agent's valuation function can be represented as single value. The task price is calculated by finding the critical value, the minimum task price required for the task to still allocated to a server. This has been shown to be a strategyproof [12] 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's 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 greedy algorithm for all task (excluding the critical task) with time complexity, $O(|J|\,|I|)$, till the critical task can no longer to allocated (a constant time function). Where the task's critical value is calculated, a 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. □

4.2.2 Critical Value Auction Strategyproof. In order that the auction is strategyproof, the value density function must be monotonic [12] 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 α 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. □

4.3 Decentralised Iterative Auction

In some application 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 [4, 6, 14]. VCG auctions calculates the price of an item by finding the difference in social welfare if the item exists and 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 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 of 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 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.

Algorithm 2 Decentralised Iterative Auction

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

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

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

4.3.1 Server revenue optimisation problem. To find the optimal revenue for a server m given a new task n' and 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 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 of the tasks plus the new task. As this formulation only considers a single server, the task allocation constraint is not consider.

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 attribute 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 till server becomes full, because of this initially 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 is 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 requirementsfrom 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 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.

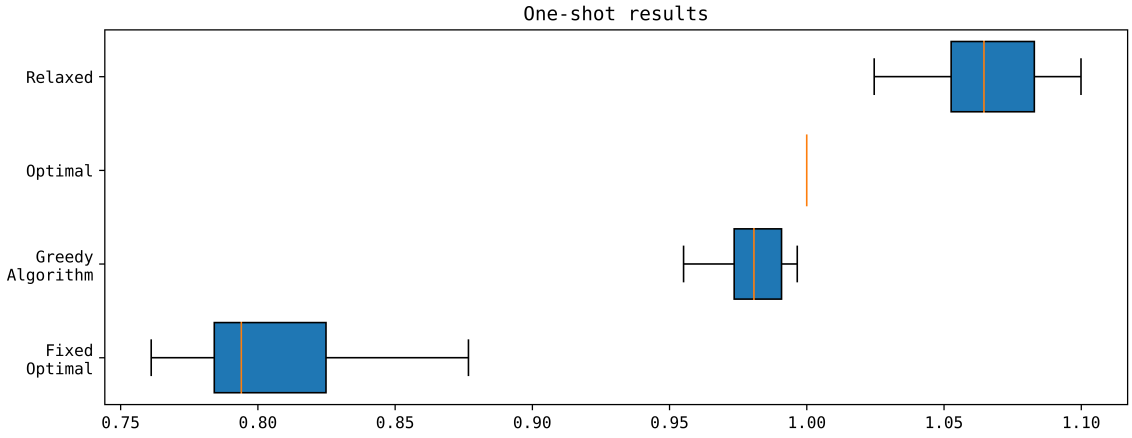


Fig. 5. Comparison of the social welfare for the greedy mechanism, optimal, relaxed problem, time limited branch and bound

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.

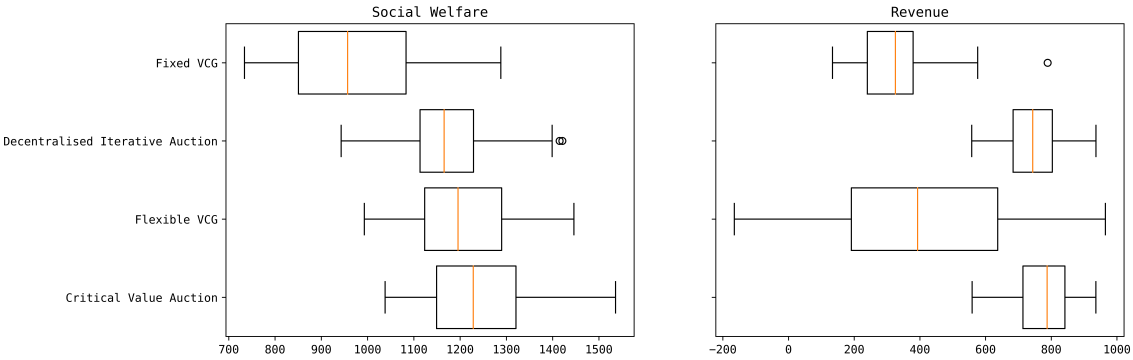
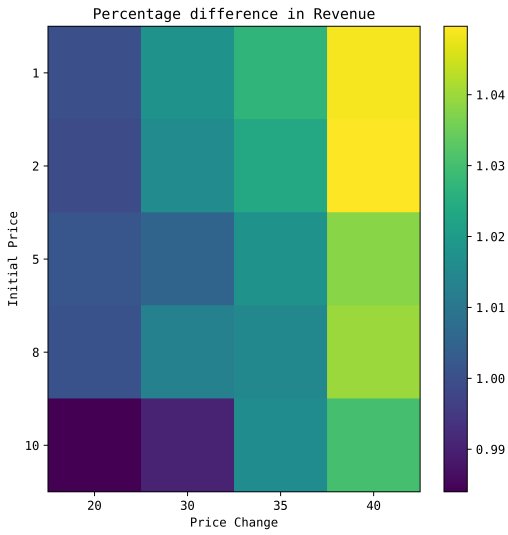


Fig. 6. Comparison of the social welfare for the auction mechanisms

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



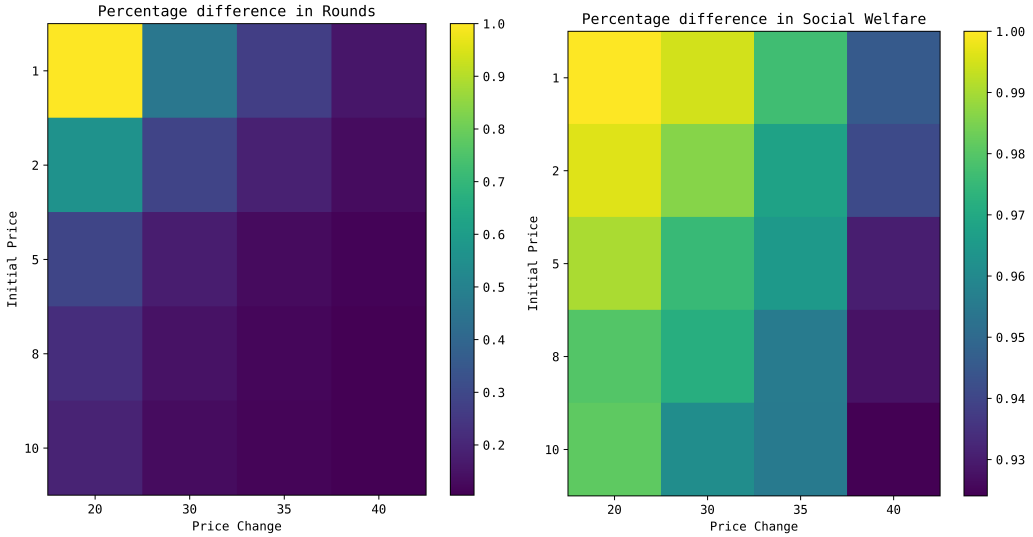


Fig. 7. Grid search of difference server price change and task initial cost

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, 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

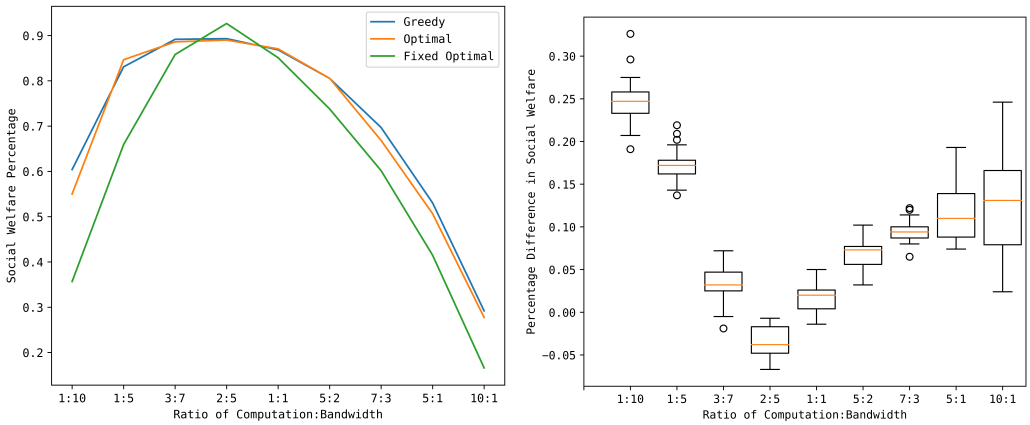


Fig. 9. Social welfare

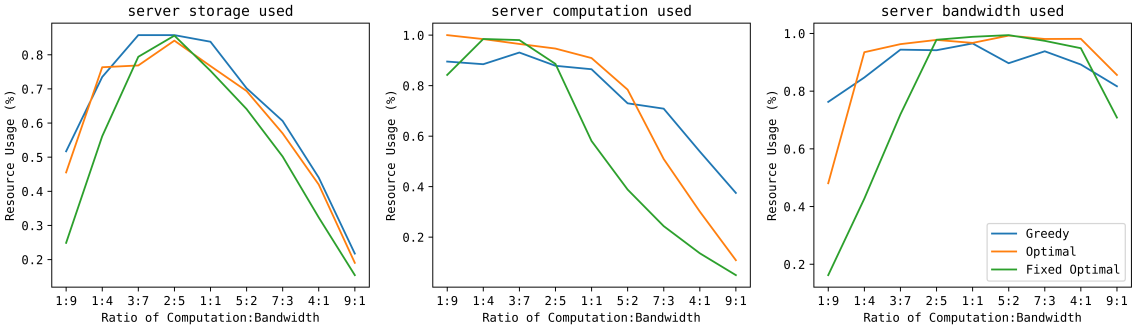


Fig. 10. Server resource usage

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] 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

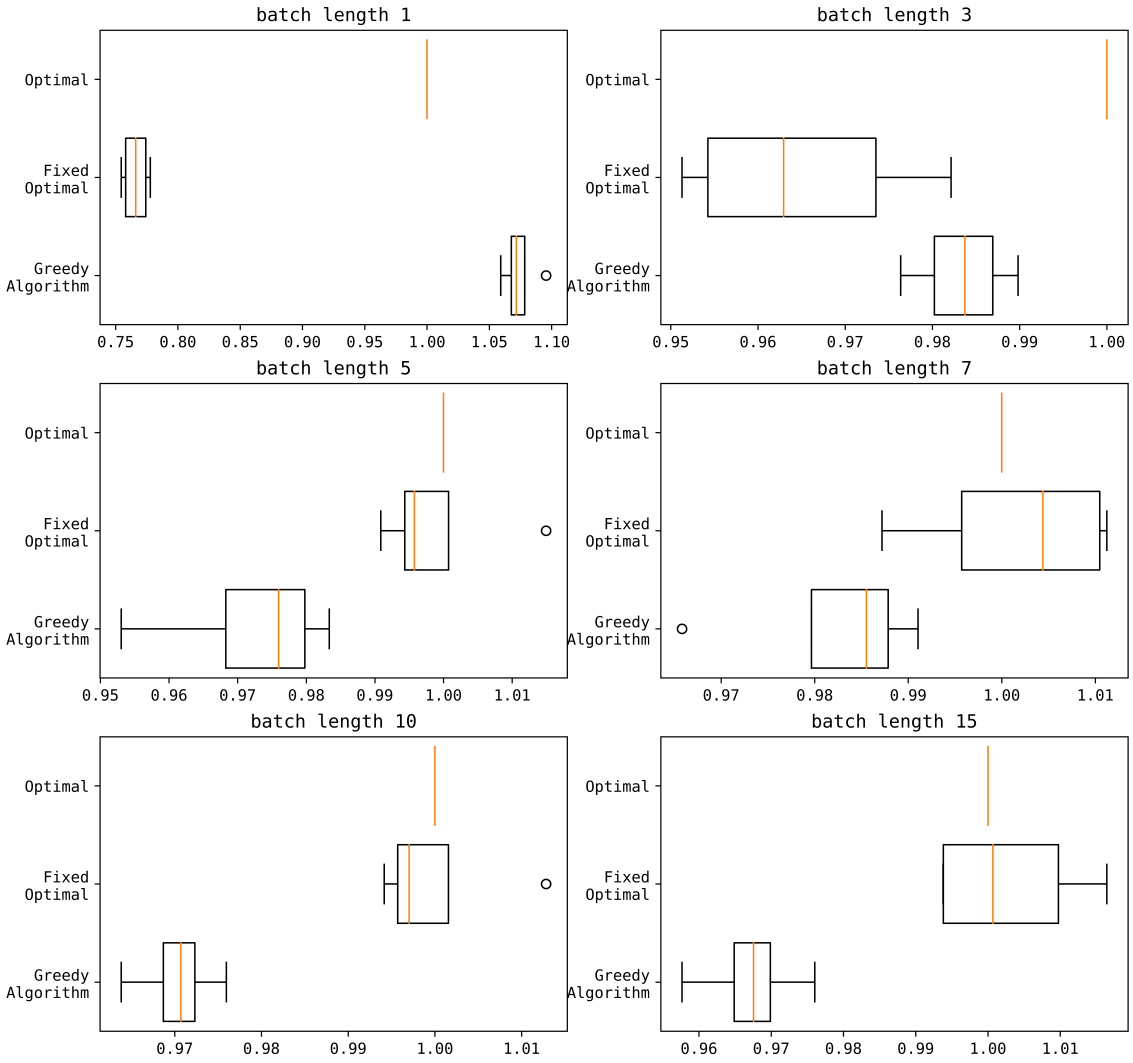


Fig. 11. Online batch lengths

[2] 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/

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

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

[5] 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

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

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

[8] 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

[9] 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).

[10] 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

[11] 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

[12] 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

[13] 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

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

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

APPENDIX

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.