Auction-based Mechanisms for Allocating Elastic Resources in Edge Clouds

Paper #1263

ABSTRACT

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

KEYWORDS

Edge clouds; elastic resources; auctions

1 INTRODUCTION

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

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

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

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

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

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

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

https://doi.org/doi

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

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

2 RELATED WORK

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

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

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

3 PROBLEM FORMULATION

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

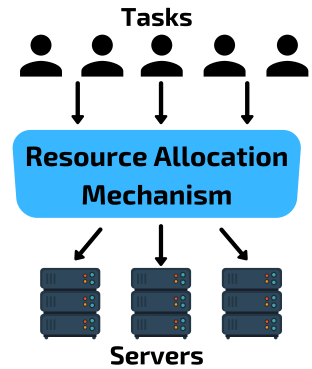

Figure 1: System Model

3.1 System model

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

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

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

3.2 Optimization problem

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

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

s.t.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

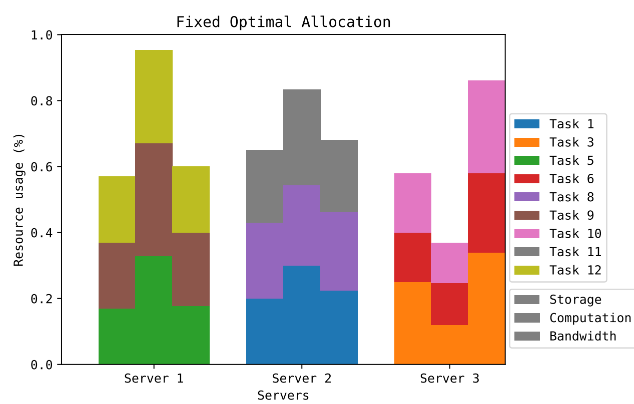

Figure 2: Optimal solution with fixed resources. Due to not being able to balance out the resources, bottlenecks on the server 1 and 2's computation have occurred

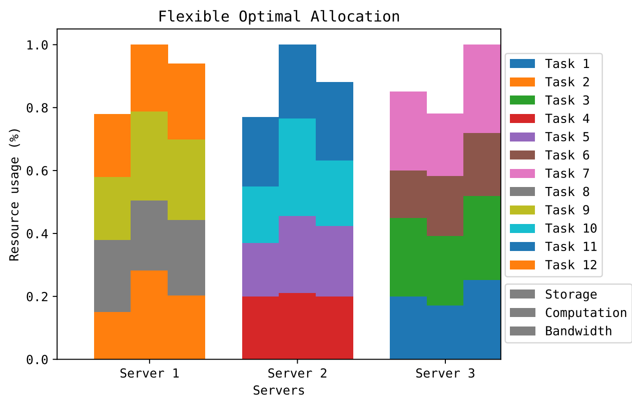

Figure 3: Optimal solution with elastic resources. Compared to the fixed allocation, the elastic allocation is able to fully use all of its resources

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

Table 1: Servers - Table of server attributes

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

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

4 FLEXIBLE RESOURCE ALLOCATION MECHANISMS

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

4.1 Greedy Mechanism

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

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

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

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

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

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

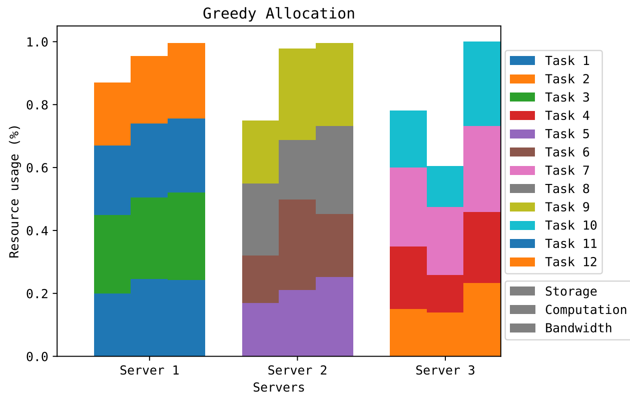

Figure 4: Example Greedy allocation using model from table 2 and 1

Algorithm 1 Greedy Mechanism

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

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

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

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

4.2 Critical Value Auction

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

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

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

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

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

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

4.3 Decentralised Iterative Auction

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

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

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

4.3.1 Server problem case.

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

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

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

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

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

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

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

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

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

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

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

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

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

Algorithm 2 Decentralised Iterative Auction

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

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

4.4 Attributes of proposed algorithms

In table 3, the important attributes for the proposed algorithm

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

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

5 EMPIRICAL EVALUATION

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

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

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

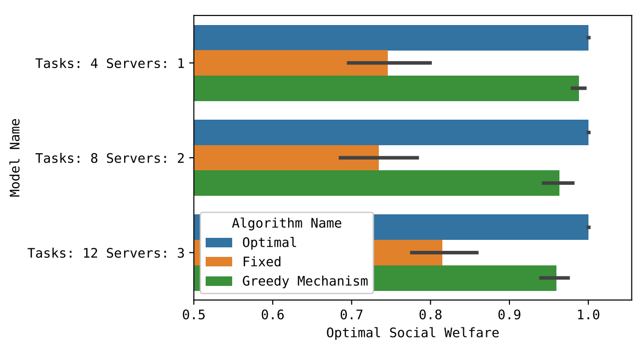

Figure 5: Comparison of the social welfare for the greedy mechanism, optimal, relaxed problem, time limited branch and bound

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

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

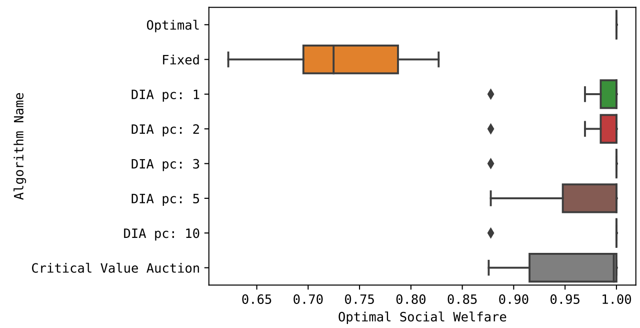

Figure 6: Comparison of the social welfare for the auction mechanisms

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

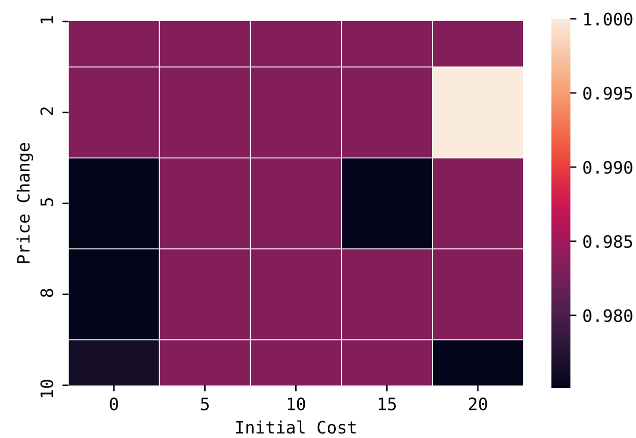

Figure 7: Average number of rounds with a price change variables and task initial cost

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

6 CONCLUSIONS

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

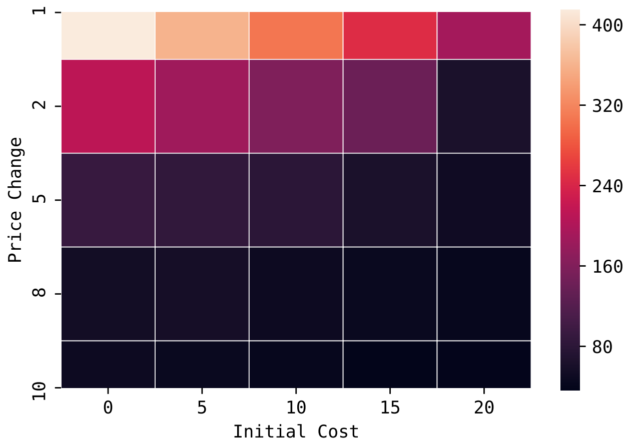

Figure 8: Average social welfare with a price change variables and task initial cost

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

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

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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