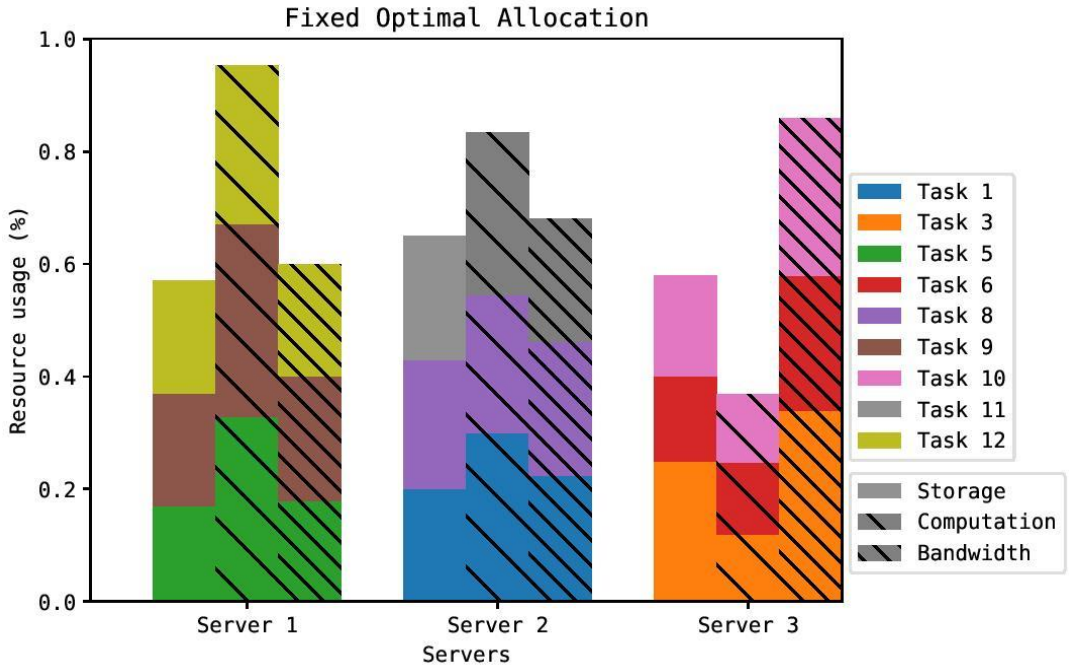
# Analytical agility at the edge of the network through auction mechanisms

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

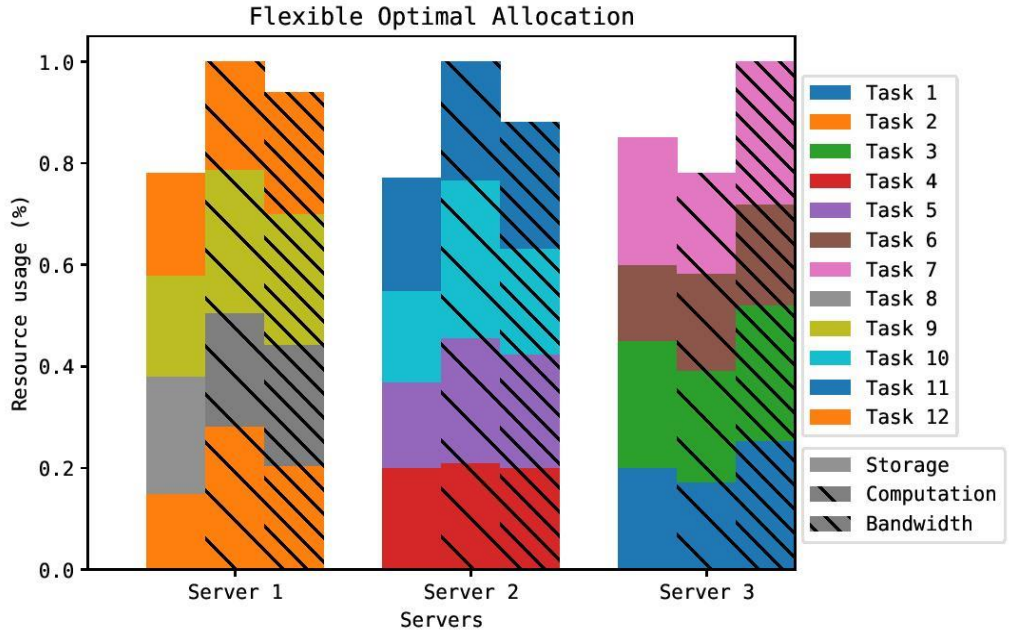
# Motivation

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




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

# Flexible resource allocation mechanism




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

## Deadline Constraint

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

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

# Optimisation problem

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

s.t.

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

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

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

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

$$0 \leq s_j' \leq \infty, \quad \forall j \in J, \quad (6)$$

$$0 \leq w_j' \leq \infty, \quad \forall j \in J, \quad (7)$$

$$0 \leq r_j' \leq \infty, \quad \forall j \in J, \quad (8)$$

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

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

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

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

# Approaches

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

# Greedy mechanism

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

**Algorithm 1** Greedy Mechanism

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

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

# Critical value auction

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

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

# Decentralised iterative auction

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

**Algorithm 2** Decentralised Iterative Auction

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

# Modified server optimisation problem

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

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

s.t.

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

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

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

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

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

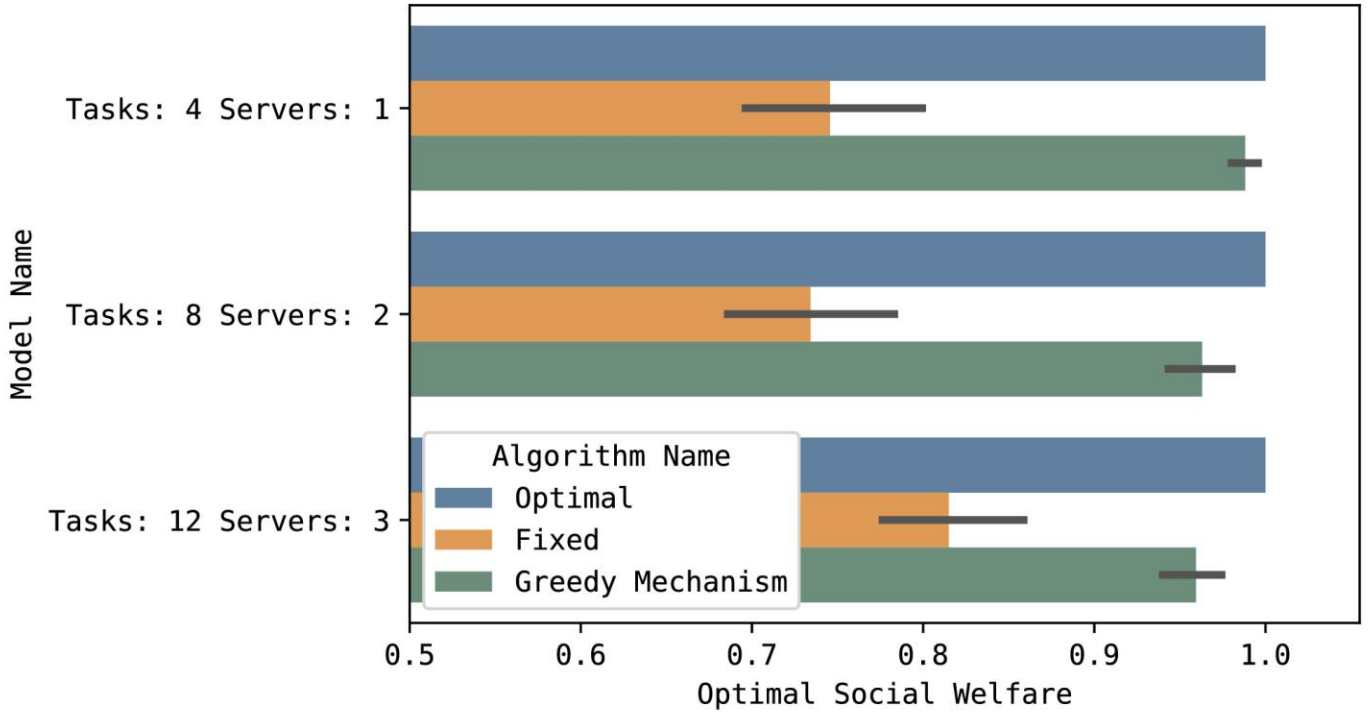
$$0 < w_n' < \infty, \qquad \forall n \in N \cup \{n'\} \tag{17}$$

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

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

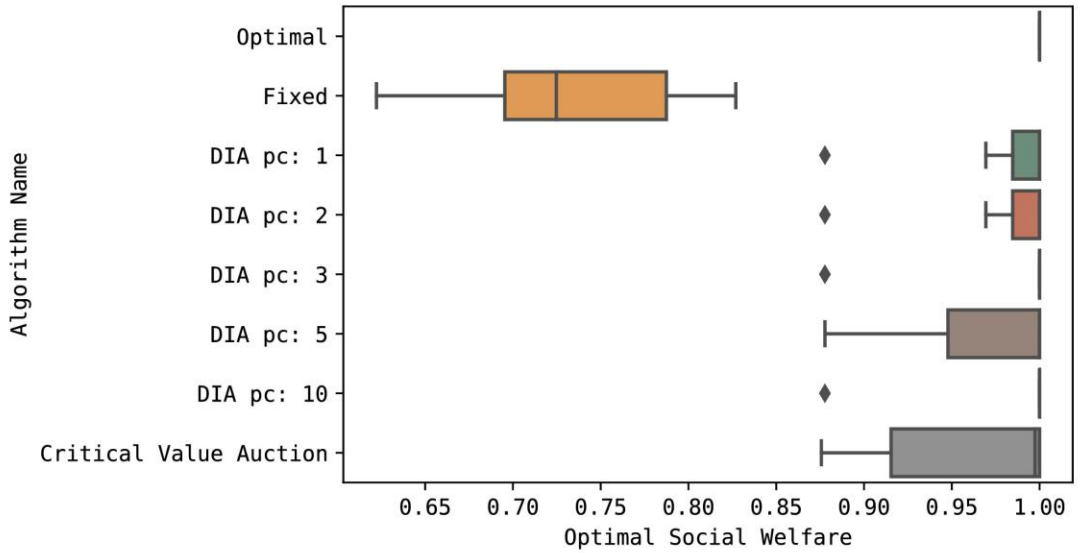
# Greedy mechanism and critical value auction results

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



# Decentralised Iterative auction

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



# Conclusion and future work

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