Auctions for Allocation of Elastic Resources in Cloud Computing

Paper #XXX
Mark Towers
mt5g17@soton.ac.uk

ABSTRACT

TODO Abstract goes here

KEYWORDS

Edge clouds, elastic resources, auctions.

1 INTRODUCTION

In the last years, cloud computing [] has become a very popular solution to run data-intensive applications remotely. The importance of running different highly-computational tasks from a remote entity became even more popular with the emerging technology of *mobile edge computing*, which enables mobile users to run delay-sensitive computationally-intensive tasks from the edge of mobile networks at small data-centers, known as *edge clouds*.

These kinds of settings are very important for military tactical network where personal on the ground do not have the capability's to run task that a server can. The application of this could be analysis of video from troops on the ground, surveillance footage from UAVs or data analysis of captured enemy electronics. To do this, several types of resource of the server and the task are considered: the communication bandwidth, computational and data storage / memory used in computing and each will have difference amounts depend on the task. As well the resources, each task will have a different importance depend on who is requesting it and what the task will achieve. For example, a general's task will be more important that a captains or analysis of a high value target's electronics is more important to be run than analysis of base CCTV cameras. In this work we demonstrate, a centralised greedy algorithm to maximise social welfare and two auction algorithms with one centralised and the other an decentralised iterative auction.

In previous work, user would request a fix amount of resource that would be used for process the task however this can create a bottleneck with certain resources particular with small servers used in edge cloud computing. In this work, task will instead request the total resources required to process the task allowing the cloud provider to distribute it's resources more efficiently to it's task due to being aware of it's tasks requirements.

2 RELATED WORK

In [**?**], the authors consider the servers to be edge clouds responsible for both deciding where to place the code/data needed to run a specific task, and also for scheduling different tasks to different edge clouds. The goal there is to maximize the average rate of successfully accomplished tasks over time. The resources provided by different edge clouds are heterogeneous and there is a monetary constraint for every task.

3 SYSTEM MODEL

A sketch of the system is illustrated in Fig. **??**. We assume that in the system there are I servers, which could be edge clouds, and could be accessed either through base stations or WiFi AP . Servers have different types of resources, such as: storage for the code/data needed to run a task, computation capacity in terms of CPU cycles needed to run a task, and communication capacity to receive the request by a task and to send back the results after the latter is executed. We assume that the servers are heterogeneous in all their characteristics. We denote the storage capacity of server i with S_i, computation capacity with W_i, whereas the communication capacity with R_i.

There are J different tasks (jobs)[1] from different entities that need to be run on those servers. Running any of these tasks on the server requires storing the appropriate code/data on the same server. These could be for example images of objects or CNN layers in identification tasks. The size of task j is denoted as s_j. A task also requires some CPU cycles from the server. This is denoted as w_j, and is expressed in the total number of flops. A related parameter to this is the rate at which the number of flops are assigned from the server per unit of time. We denote this as $w_{i,j}$, and its unit is Mflops/s. Finally, after the task is run and the results obtained, the latter need to be sent back to the user. The size of these results is denoted with r_j, and the rate at which they are sent from server i (if the task was executed there) to the user is $r_{i,j}$. As opposed to the storage requirement of the task s_j that is *strict*, the computation and communication parameters $w_{i,j}$ and $r_{i,j}$ are *elastic*, i.e., the job can be run under different values. The values of these parameters would be constrained by the deadline D_j that any task is assigned. It is the maximum time the time is allowed to "spend" in the system. If the task is executed completely within that time interval by any of the servers, then the task gets the full utility U_j, which is specific for any task. In case the deadline is not met, then the utility is 0. So, we have *all* or *nothing* task execution scheme. It should also be mentioned that since the resources are finite, each task can be assigned to at most one server, and once assigned it cannot switch to another server.

In this paper, we consider the static case scenario, where all jobs arrive at the same time at the start of the program. The dynamic case (where task arrive over time) is beyond the scope of this paper. This is no task preemption as all jobs run concurrently and so cant be stopped. We assume that every task will request at least as much

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

[1] In this paper, we will use the notions job and task interchangeably.

resources as required to run as any program would fail if resources were under reported.

$$\max \sum_{j}^{J} U_j \left(\sum_{i}^{I} x_{i,j}\right) \tag{1}$$

$$\text{s.t.} \tag{2}$$

$$\sum_{j=1}^{J} S_j x_{i,j} \leq S_i, \qquad \forall i \in I, \tag{3}$$

$$\sum_{j=1}^{J} w_j x_{i,j} \leq W_i, \qquad \forall i \in I, \tag{4}$$

$$\sum_{j}^{J} (r_j + s_j) x_{i,j} \leq R_i, \qquad \forall i \in I, \tag{5}$$

$$\frac{S_j}{s_j} + \frac{W_j}{w_j} + \frac{R_j}{r_j} \leq \frac{D_j}{x_{i,j}}, \qquad \forall i \in I, j \in J, \tag{6}$$

$$s_{i,j} > 0, \qquad \forall i \in I, j \in J, \tag{7}$$

$$w_{i,j} > 0, \qquad \forall i \in I, j \in J, \tag{8}$$

$$r_{i,j} > 0, \qquad \forall i \in I, j \in J, \tag{9}$$

$$\sum_{i=1}^{I} x_{i,j} \leq 1, \qquad \forall j \in J, \tag{10}$$

$$x_{i,j} \in \{0, 1\}, \qquad \forall i \in I, j \in J \tag{11}$$

The objective (Eq.(??)) is to maximize the total utility over all tasks. Task j will receive the full reward U_j only if the task is executed entirely and the results are obtained within the deadline for that task. Constraint (Eq.(??)) relates to the finite storage capacity of every server to store code/data to run tasks. The finite computation capacity of every server is expressed through Eq.(??), whereas Eq.(??) denotes the constraint on the communication capacity of the servers. The communication bandwidth comprises of two parts: to send the data/code or request to the server, and to get the results back to the user. Constraint Eq.(??) is the deadline associated with every task, where the total time of 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. The rates at which the code is sent, run and the results are sent back are all non-negative (Eqs(??))(??))). Further, every task is served by at most one server Eq.(??)). Finally, a task is either served or not. Hence, the binary nature of $x_{i,j}$ (Eq.(??)).

NP-hardness: This optimization problem is a more general case of the 0-1 knapsack problem, which is known to be NP-hard []. As a result, this is an NP-hard problem. In the next section, we propose a heuristic, with a performance guarantee of $\frac{1}{n}$.

4 FLEXIBLE RESOURCE ALLOCATION MECHANISMS

Due to the flexibility of the amount of resource allocated to each task, this has meant that Dynamic Programming [] and Bin Packing algorithms for Multiple Knapsack Problems [] cant to use as they require the weights to be fixed. Non-linear integer programming is possible to use to solve this problem however the problems hugely intractable with larger model sizes then 15 jobs and 3 servers.

Combinatorial auction are a class of auction that allow for multiple resources to be sold and brought in a single bid however due to flexible nature of the problem case then these algorithms are largely unable to be used.
Because of this, we have implemented a critical value auction using the Greedy Mechanism that is known to be strategyproof that is centralised and requires all users to reveal their private values. And a second auction, that is a decentralised iterative auction using a reverse VCG mechanism to calculate the prices.

4.1 Greedy Mechanism

Greedy mechanism is a three stage algorithm where the jobs are ranked used a value density function. Then each jobs from the sorted job ranks are assigned to a server using a server selection function and then the resource are allocated to the

4.1.1 Greedy algorithm. The greedy algorithm code in python

```
ranked_jobs = sort(jobs, key=lambda j: value_density(j))
for job in ranked_jobs:
    allocated_server = server_selection(job, servers)

    if allocated_server:
        loading_speed, compute_speed, sending_speed =
            ↪ resource_allocation(job, allocated_server)
        allocate(job, allocated_server, loading_speed, compute_speed,
            ↪ sending_speed)
```

4.1.2 Lower bound of greedy mechanism. Due to the flexible nature of jobs then when allocating jobs, to be done optimally would involved being able to know the future allocations and then plan this jobs based on that. However this is not possible to do there the algorithm can only guarantee that a single job can be allocated when the jobs are ranked purely by the utility of the job.

4.2 Decentralised Iterative Auction

The VCG auction is a well known auction that is known to be economically efficient, budget balanced and incentive compatible by finding the price of a job by calculating the social welfare if the job didnt exist compared to when it does exist. Job are then evaluated on how good it is for the rest of the other jobs to exist and just requires a integer programming program which works for our problem.
our iterative auction uses the same idea of calculating the price based on the how its allocation affects the rest of the jobs. This is done through calculating through calculating the total revenue that a server will receive if the new job is running for 0 and has the option of allocating all of the current job at their current price. The price of the new job is equal to the difference of the old allocation revenue and the new allocation revenue (being the price that the new job will have to offer so that the server will have as much profit) plus a small value to make the server more revenue.
Because of this formulation, the server will solve a slightly different linear programming problem from the optimal algorithm.

$$\max \sum_{j}^{J} P_j(\sum_{i}^{I} x_{i,j}) \tag{12}$$

s.t. (13)

$$\sum_{j}^{J} S_j x_{i,j} + S_k \leq S_i, \qquad \forall i \in I, \tag{14}$$

$$\sum_{j}^{J} w_j x_{i,j} + w_k \leq W_i, \qquad \forall i \in I, \tag{15}$$

$$\sum_{j}^{J} (r_j + s_j) x_{i,j} + (r_k + s_k) \leq R_i, \qquad \forall i \in I, \tag{16}$$

$$\frac{s_j}{s_{i,j}} + \frac{w_j}{w_{i,j}} + \frac{r_j}{r_{i,j}} \leq \frac{D_j}{x_{i,j}}, \qquad \forall i \in I, j \in J \cup \{k\}, \tag{17}$$

$$s_{i,j} > 0, \qquad \forall i \in I, j \in J \cup \{k\} \tag{18}$$

$$w_{i,j} > 0, \qquad \forall i \in I, j \in J \cup \{k\} \tag{19}$$

$$r_{i,j} > 0, \qquad \forall i \in I, j \in J \cup \{k\} \tag{20}$$

$$\sum_{i=1}^{I} x_{i,j} \leq 1, \qquad \forall j \in J, \tag{21}$$

$$x_{i,j} \in \{0, 1\}, \qquad \forall i \in I, j \in J \tag{22}$$

4.2.1 Decentralised Iterative auction properties. TODO

4.3 Critical Value Auction

Single property domain auctions [**?**] allow for strategyproof (weakly-dominant incentive compatible) auction that uses an allocation algorithm like **??** that the job value is reduced till the job wouldn't be allocated.

4.3.1 Critical Value algorithm. The critical value algorithm that uses a custom binary search to find the point that the job is no longer allocated

```
ranked_jobs = sort(jobs, key=lambda j: value_density(j))
greedy_algorithm(ranked_jobs, servers)

for job in [job for job in ranked_jobs if job.allocated]:
    lower_bound = ranked_jobs.index(job)
    upper_bound = len(ranked_jobs) − 1
    jobs = ranked_jobs.copy()

    while lower_bound < upper_bound:
        pos = floor((lower_bound + upper_bound) / 2)
        jobs.remove(job)
        jobs.insert(pos, job)

        greedy_algorithm(jobs, servers)
        if job.allocated:
            lower_bound = pos + 1
        else:
            upper_bound = pos − 1

    if lower_bound == len(ranked_jobs) − 1:
        return 0
    else:
        ranked_jobs[lower_bound].value
```

5 EMPIRICAL EVALUATION

6 CONCLUSIONS AND FUTURE WORK

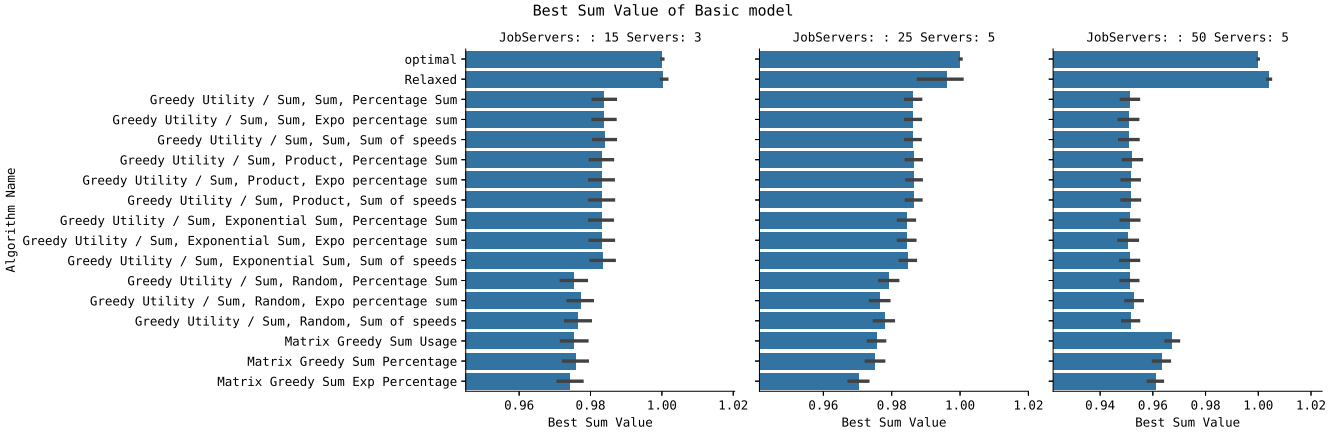

Figure 1: Greedy algorithms with basic model measuring the sum of values

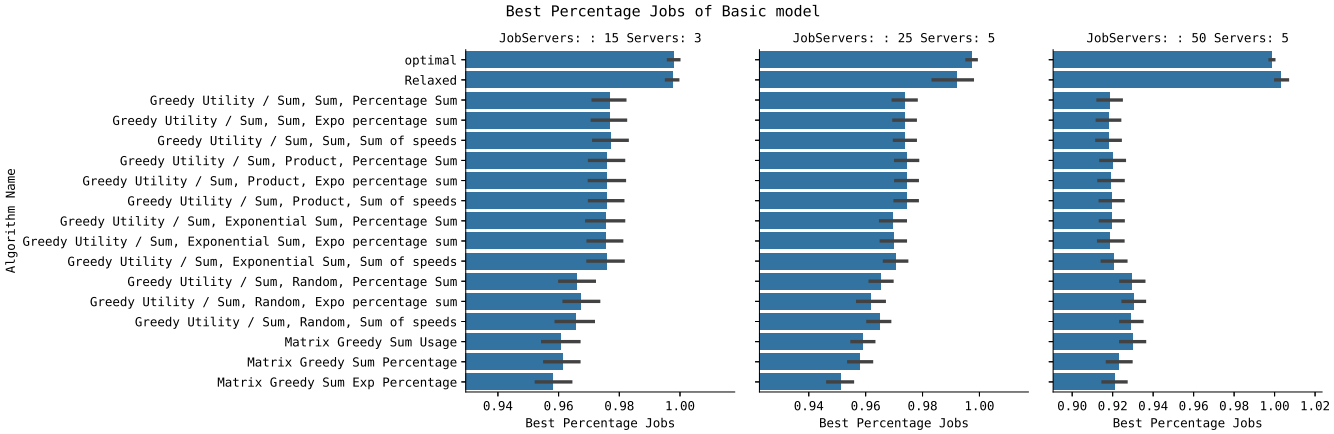

Figure 2: Greedy algorithms with basic model measuring the percentage of jobs allocated

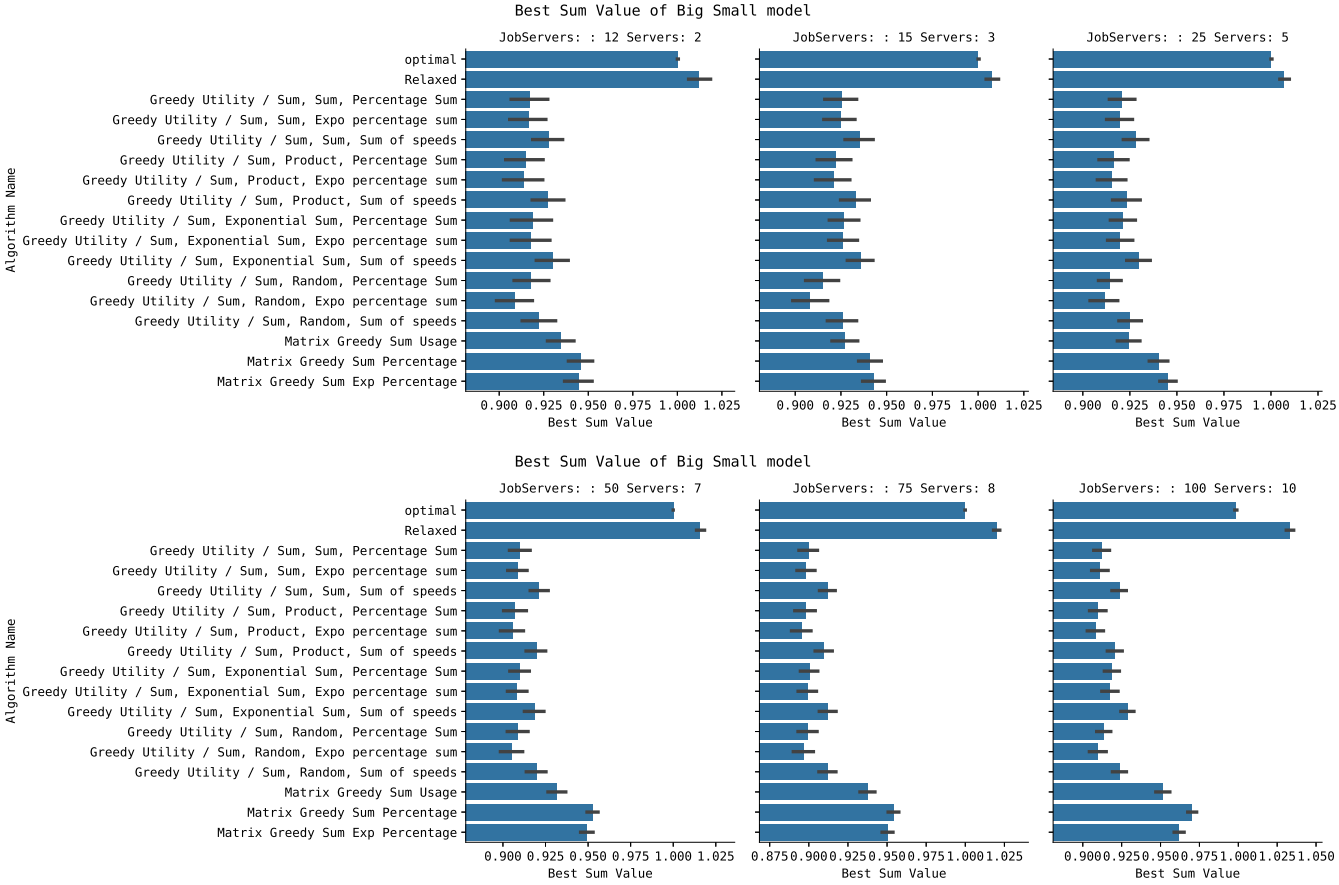

Figure 3: Greedy algorithms with big small model measuring the sum of values

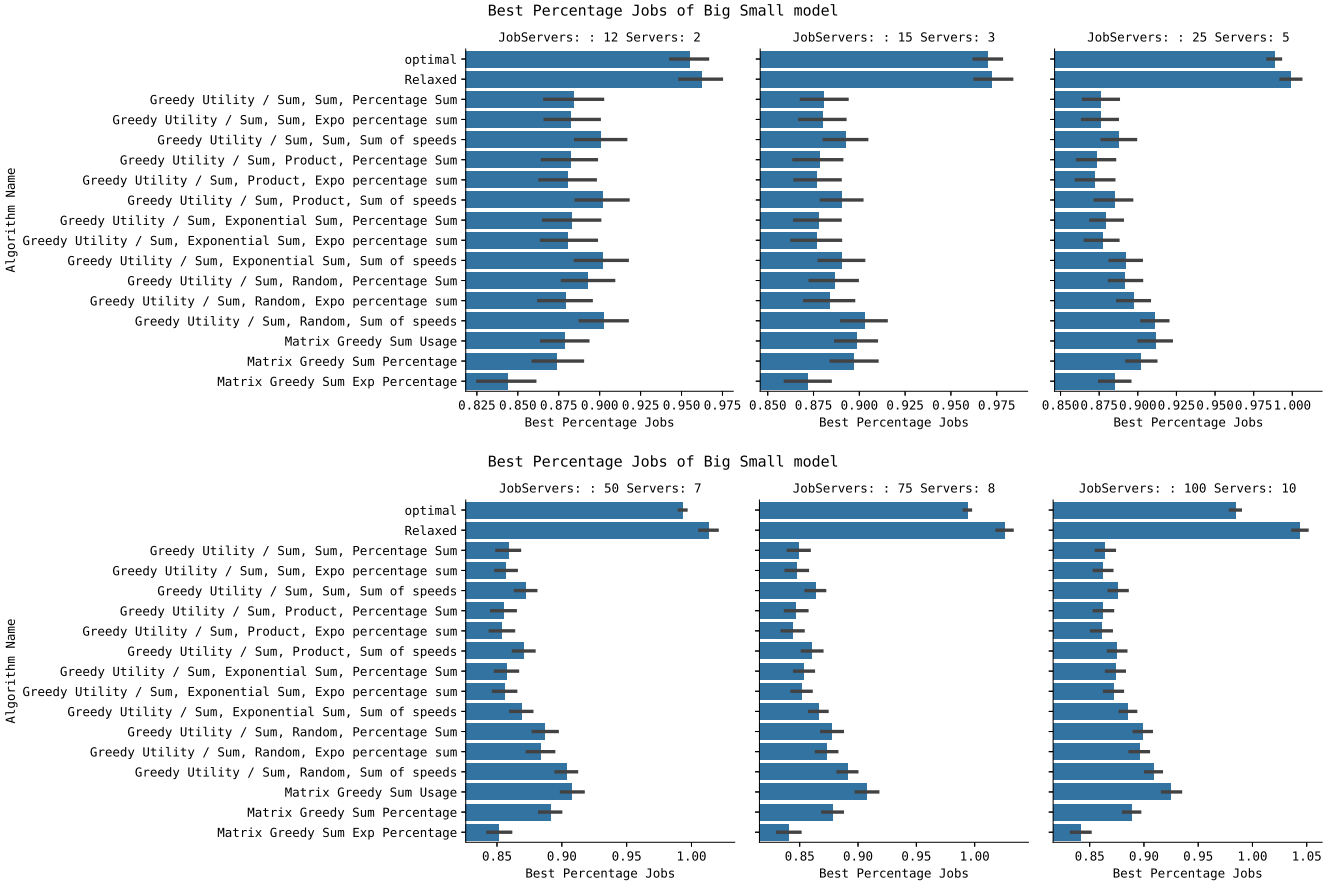

Figure 4: Greedy algorithms with big small model measuring the percentage of jobs allocated

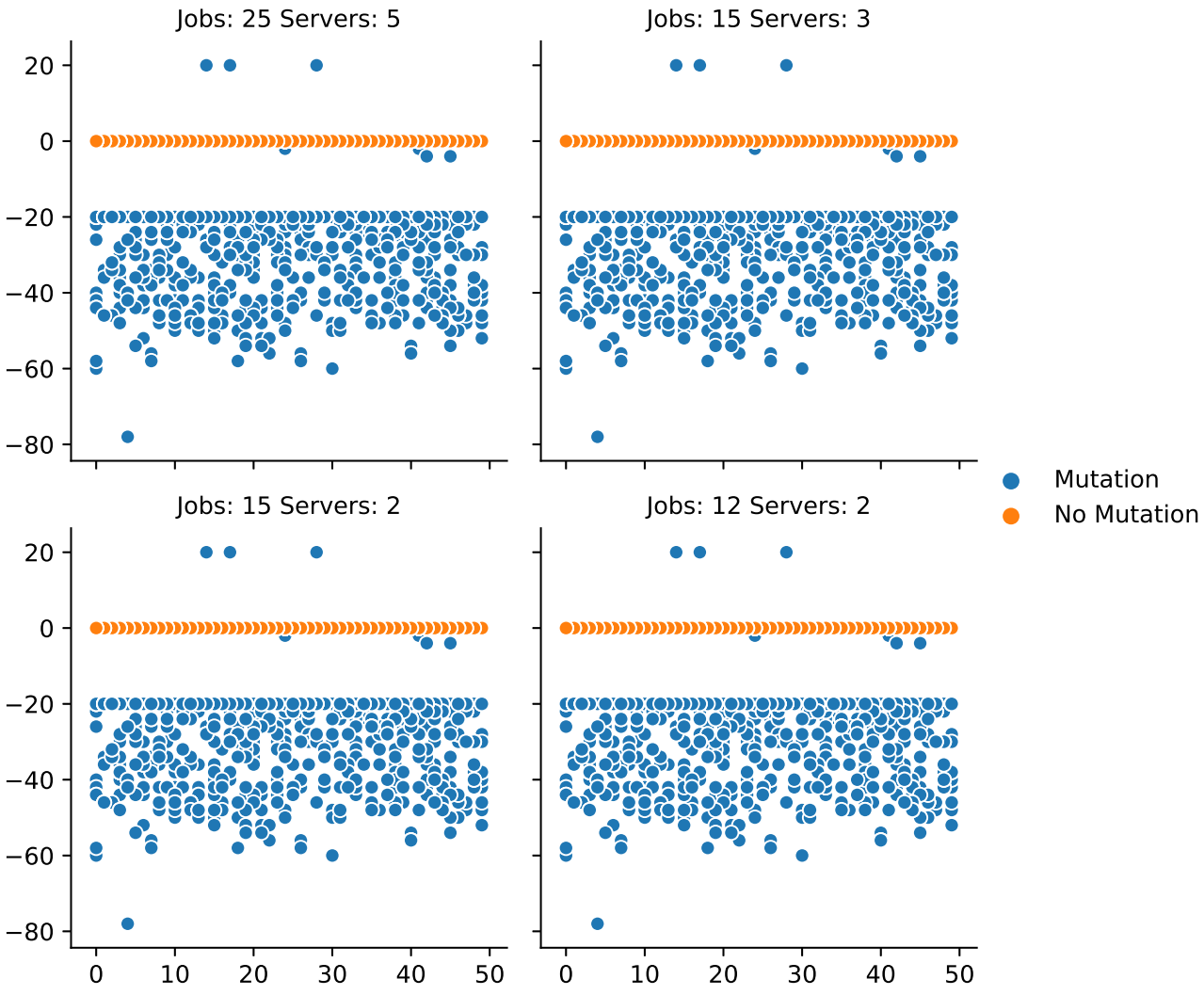

Figure 5: Measuring the difference in price when the job is incorrectly reported

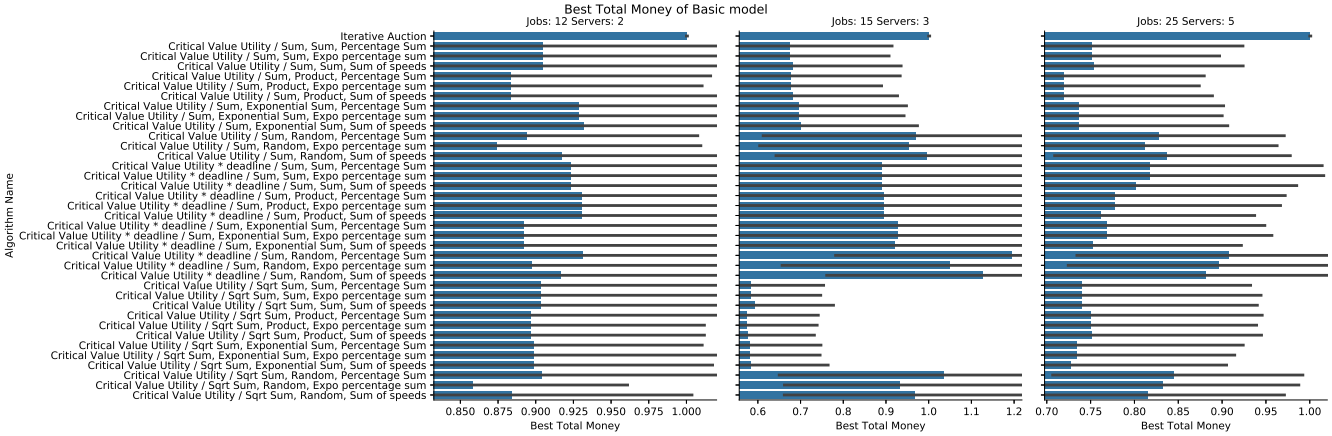

Figure 6: The results from the critical value algorithm total price