A Truthful $(1-\varepsilon)$-Optimal Mechanism for On-demand Cloud Resource Provisioning

Xiaoxi Zhang $^1$, Chuan Wu $^1$, Zongpeng Li $^2$, Francis C.M. Lau $^1$

$^1$ Department of Computer Science, The University of Hong Kong
$^2$ Department of Computer Science, University of Calgary
Outline

Background

Why Auction

How our mechanism works

Challenges
Background

- Cloud providers provide computing resources distributed in multiple data centers
- Users are geo graphically distributed
- Users want various types of resources
○ Background

* Virtualization technology packs resources into VMs
Background

* Pre-determined types

<table>
<thead>
<tr>
<th>Model</th>
<th>vCPU</th>
<th>Mem (GiB)</th>
<th>SSD Storage (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c3.large</td>
<td>2</td>
<td>3.75</td>
<td>2 x 16</td>
</tr>
<tr>
<td>c3.xlarge</td>
<td>4</td>
<td>7.5</td>
<td>2 x 40</td>
</tr>
<tr>
<td>c3.2xlarge</td>
<td>8</td>
<td>15</td>
<td>2 x 80</td>
</tr>
<tr>
<td>c3.4xlarge</td>
<td>16</td>
<td>30</td>
<td>2 x 160</td>
</tr>
<tr>
<td>c3.8xlarge</td>
<td>32</td>
<td>60</td>
<td>2 x 320</td>
</tr>
</tbody>
</table>

vCPU: 16
Mem: 16 GB
Data: 160GB
Background

- Pre-determined types

vCPU: 16
Mem: 16 GB
Data: 160GB

<table>
<thead>
<tr>
<th>Model</th>
<th>vCPU</th>
<th>Mem (GiB)</th>
<th>SSD Storage (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c3.large</td>
<td>2</td>
<td>3.75</td>
<td>2 x 16</td>
</tr>
<tr>
<td>c3.xlarge</td>
<td>4</td>
<td>7.5</td>
<td>2 x 40</td>
</tr>
<tr>
<td>c3.2xlarge</td>
<td>8</td>
<td>15</td>
<td>2 x 80</td>
</tr>
<tr>
<td><strong>c3.4xlarge</strong></td>
<td><strong>16</strong></td>
<td><strong>30</strong></td>
<td><strong>2 x 160</strong></td>
</tr>
<tr>
<td>c3.8xlarge</td>
<td>32</td>
<td>60</td>
<td>2 x 320</td>
</tr>
</tbody>
</table>
**Current Development**

- More VM types
- Fixed pricing

<table>
<thead>
<tr>
<th>VM Type</th>
<th>vCPUs</th>
<th>Memory</th>
<th>Price per Hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>c4.large</td>
<td>2</td>
<td>8</td>
<td>$0.116</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>16</td>
<td>$0.232</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>31</td>
<td>$0.464</td>
</tr>
<tr>
<td>c4.4xlarge</td>
<td>16</td>
<td>62</td>
<td>$0.928</td>
</tr>
<tr>
<td>c4.8xlarge</td>
<td>36</td>
<td>132</td>
<td>$1.856</td>
</tr>
<tr>
<td>c3.large</td>
<td>2</td>
<td>7</td>
<td>$0.105</td>
</tr>
<tr>
<td>c3.xlarge</td>
<td>4</td>
<td>14</td>
<td>$0.210</td>
</tr>
<tr>
<td>c3.2xlarge</td>
<td>8</td>
<td>28</td>
<td>$0.420</td>
</tr>
<tr>
<td>c3.4xlarge</td>
<td>16</td>
<td>55</td>
<td>$0.840</td>
</tr>
<tr>
<td>c3.8xlarge</td>
<td>32</td>
<td>108</td>
<td>$1.680</td>
</tr>
</tbody>
</table>
- How many VM types do we need?
  - Difficult to estimate since users’ demands are widely different
  - Users want customized VM types

- Under fixed pricing for each type of VMs, it is impossible to:
  - come up with the appropriate prices
  - maximize the social welfare
Why Auction

* How many VM types do we need?
  -- Difficult to estimate since users’ demands are widely different
  -- Users want customized VM types

How to address it?

✧ Users determine their own VM types

* Under fixed pricing for each type of VMs, it is difficult:
  -- come up with the appropriate prices
  -- maximize the social welfare

How to address it?

✧ Providers price according to the current demand and supply relationship
Our Auction Model

\* Model:
N users, I bids
(Each user submits as many bids as he wishes with at most one accepted)
K types of resources
D data centers (capacities known)
Bidding Language: \( B_i = (b_i, \{R_{kd}\}) \)

Achieves:
\* (1-\(\varepsilon\))-optimal social welfare
\* Polynomial running time in expectation
\* Truthfulness in expectation
Achieves:

* (1-ε)-optimal social welfare
* Polynomial running time in expectation

Allocation is An NP-hard combinatorial optimization problem

* Truthfulness in expectation

VCG requires an exact optimal allocation
○ How Our Mechanism Works

* VM Allocation:

\[
\text{maximize} \quad \sum_{n \in [N]} \sum_{i \in \mathcal{B}_n} b_i x_i
\]

subject to:

\[
\sum_{n \in [N]} \sum_{i \in \mathcal{B}_n} x_i R^i_{k,d} \leq c_{k,d}, \quad \forall k \in [K], \forall d \in [D],
\]

\[
\sum_{i \in \mathcal{B}_n} x_i \leq 1, \quad \forall n \in [N],
\]

\[
x_i \in \{0, 1\}, \quad \forall i \in \mathcal{B}_n, \forall n \in [N].
\]
How Our Mechanism Works

VM Allocation:

\[
\begin{align*}
\text{maximize} & \quad \sum_{n \in [N]} \sum_{i \in \mathcal{B}_n} b_i x_i \\
\text{subject to:} & \quad \sum_{n \in [N]} \sum_{i \in \mathcal{B}_n} x_i R_{k,d}^i \leq c_{k,d}, \quad \forall k \in [K], \forall d \in [D], \\
& \quad \sum_{i \in \mathcal{B}_n} x_i \leq 1, \quad \forall n \in [N], \\
& \quad x_i \in \{0, 1\}, \quad \forall i \in \mathcal{B}_n, \forall n \in [N].
\end{align*}
\]
How Our Mechanism Works

VM Allocation:

\[
\begin{align*}
\text{maximize} & \quad \sum_{n \in [N]} \sum_{i \in B_n} b_i x_i \\
\text{subject to:} & \quad \sum_{n \in [N]} \sum_{i \in B_n} x_i R_{k,d}^i \leq c_{k,d}, \quad \forall k \in [K], \forall d \in [D], \\
& \quad \sum_{i \in B_n} x_i \leq 1, \quad \forall n \in [N], \\
& \quad x_i \in \{0, 1\}, \quad \forall i \in B_n, \forall n \in [N].
\end{align*}
\]
How Our Mechanism Works

VM Allocation:

\[
\text{maximize} \quad \sum_{n \in [N]} \sum_{i \in B_n} b_i x_i
\]

subject to:

\[
\sum_{n \in [N]} \sum_{i \in B_n} x_i R^i_{k,d} \leq c_{k,d}, \quad \forall k \in [K], \forall d \in [D],
\]

\[
\sum_{i \in B_n} x_i \leq 1, \quad \forall n \in [N],
\]

\[
x_i \in \{0, 1\}, \quad \forall i \in B_n, \forall n \in [N].
\]
How Our Mechanism Works

VM Allocation:

maximize \[ \sum_{n \in [N]} \sum_{i \in B_n} b_i x_i \]

subject to:

- Demand: \[ \sum_{n \in [N]} \sum_{i \in B_n} x_i R_{k,d}^i \leq c_{k,d}, \quad \forall k \in [K], \forall d \in [D], \]

- Capacity: \[ \sum_{i \in B_n} x_i \leq 1, \quad \forall n \in [N], \]

- Binary: \[ x_i \in \{0, 1\}, \quad \forall i \in B_n, \forall n \in [N]. \]
How Our Mechanism Works

VM Allocation:

\[
\text{maximize} \quad \sum_{i \in B_n} \sum_{n \in [N]} b_i x_i \\
\text{subject to:}
\]

- **Demand**
  \[
  \sum_{n \in [N]} \sum_{i \in B_n} x_i R_{k,d}^i \leq c_{k,d}, \quad \forall k \in [K], \forall d \in [D],
  \]

- **Capacity**
  \[
  \sum_{i \in B_n} x_i \leq 1, \quad \forall n \in [N],
  \]

- **XOR bid constraint**
  \[
  x_i \in \{0, 1\}, \quad \forall i \in B_n, \forall n \in [N].
  \]
How Our Mechanism Works

— Big Picture

* **Optimization Model:**
  Multi-dimensional Knapsack Problem
  2-dimensional  ➔  No FPTAS  ➔  Allocation is difficult

* **Allocation:**
  Randomized Perturbation + Exact Algorithm + Sampling
  ➔ Maximal-in-distributional Range (MIDR) Mechanism

* **Payment:**  Randomized VCG
Sketch of the Allocation Algorithm

1: Perturb bidding prices in the allocation problem

2: Exactly solve the perturbed problem to obtain its optimal solution $x^p$

3: Sample allocation solution $y^\varepsilon$ following a distribution:

$$\Omega(\bar{x}^p) = \begin{cases} 
Pr[y^\varepsilon = \bar{x}^p] = 1 - \varepsilon, \\
Pr[y^\varepsilon = \bar{l}_i] = \frac{\sum_{j=1}^{L} \theta_j x_j^p}{L}, \forall i \in \{1, ..., L\}, \\
Pr[y^\varepsilon = \bar{0}] = 1 - Pr[y^\varepsilon = \bar{x}^p] - \sum_{i=1}^{L} Pr[y^\varepsilon = \bar{l}_i].
\end{cases}$$
Sketch of the Allocation Algorithm

1: Perturb bidding prices in the allocation problem
   How to perturb?

2: Exactly solve the perturbed problem to obtain its optimal solution $x^p$
   How to exactly solve it?

3: Sample allocation solution $y^\varepsilon$ following a distribution:

$$\Omega(\bar{x}^p) = \begin{cases} 
Pr[y^\varepsilon = \bar{x}^p] = 1 - \varepsilon, \\
Pr[y^\varepsilon = \bar{l}_i] = \frac{\sum_{j=1}^L \theta_j x_j^p}{L}, \forall i \in \{1, ..., L\}, \\
Pr[y^\varepsilon = \bar{0}] = 1 - Pr[y^\varepsilon = \bar{x}^p] - \sum_{i=1}^L Pr[y^\varepsilon = \bar{l}_i]. 
\end{cases}$$

Why sampling?
Sketch of the Allocation Algorithm

1: Perturb bidding prices in the Allocation Problem

How to perturb?

\[ \overline{b}_i = (1 - \epsilon)b_i + \frac{\theta_i \sum_{j=1}^{L} b_j}{L}, \forall i \in [L]. \]

Random Variables \( \{\theta_1, \theta_2, \ldots, \theta_L\} \) are independently and uniformly chosen from \( [0, \frac{\epsilon}{L}] \)

Exact Algorithm works on:

maximize \[ \sum_{n \in [N]} \sum_{i \in B_n} \overline{b}_i x_i \]
Sketch of the Allocation Algorithm

1: Perturb bidding prices in the allocation problem

How to perturb?

$$\bar{b}_i = (1 - \epsilon)b_i + \frac{\theta_i \sum_{j=1}^{L} b_j}{L}, \forall i \in [L].$$

2: Exactly solve the perturbed problem to obtain its optimal solution $x^p$

How to exactly solve it?

3: Sample allocation solution $y^\epsilon$ following a distribution:

$$\Omega(\bar{x}^p) = \begin{cases} 
Pr[y^\epsilon = \bar{x}^p] = 1 - \epsilon, \\
Pr[y^\epsilon = \bar{l}_i] = \frac{\sum_{j=1}^{L} \theta_j x_j^p}{L}, \forall i \in \{1, ..., L\}, \\
Pr[y^\epsilon = \bar{0}] = 1 - Pr[y^\epsilon = \bar{x}^p] - \sum_{i=1}^{L} Pr[y^\epsilon = \bar{l}_i].
\end{cases}$$

Why sampling?
Two Concepts of an Exact Algorithm to Solve Perturbed Problem

Pareto Optimal Solution

$\vec{x}$ is a Pareto Optimal Solution if and only if:

1. $\vec{x}$ is feasible to Allocation Problem
2. There is no a feasible solution that simultaneously achieves:
   - a larger social welfare than $\vec{x}$
   - a smaller total demand than $\vec{x}$ for each type of resource.

Pareto Optimal Set $\mathcal{P}(i)$

$\mathcal{P}(i)$ is the set of all the Pareto Optimal Solutions when considering the first $i$ bids, i.e., $\mathcal{P}(i) = \{\vec{x}^{(i)}\}$. 
Sketch of an Exact Algorithm to Solve Perturbed Problem

If first bid’s demand does not exceed resource capacity, \( P(1) = \{0, 1\} \); otherwise, \( P(1) = \{0\} \)

For \( i = 2, \ldots, L \) do:

For all \( \overrightarrow{x}^{(i-1)} \in P(i - 1) \),

1. Add 1 after the \( (i-1) \)th element of \( \overrightarrow{x}^{(i-1)} \) to get a \( \overrightarrow{x}^{(i)} \) (e.g., \( \overrightarrow{x}^{(i-1)} = 1101 \), then \( \overrightarrow{x}^{(i)} = 11011 \));
2. If \( \overrightarrow{x}^{(i)} \) is feasible, put it into \( P(i)' \).
3. Add 0 after the \( (i-1) \)th element of \( \overrightarrow{x}^{(i-1)} \) to get a \( \overrightarrow{x}^{(i)} \)
4. Merge all the \( \overrightarrow{x}^{(i)} \) and prune all the solutions that are not Pareto Optimal Solutions to get \( P(i) \)

Choose the optimal solution in \( P(L) \)

Dynamic Programming Approach
Sketch of the Allocation Algorithm

1: Perturb bidding prices in the allocation problem
   How to perturb?
   \[ \bar{b}_i = (1 - \epsilon)b_i + \frac{\theta_i \sum_{j=1}^{L} b_j}{L}, \forall i \in [L]. \]

2: Exactly solve the perturbed problem to obtain its optimal solution \( x^p \)
   How to exactly solve it? Pareto Optimal-based DP

3: Sample allocation solution \( y^\epsilon \) following a distribution:
   \[
   \Omega(\bar{x}^p) = \left\{ \begin{array}{l}
   Pr[y^\epsilon = \bar{x}^p] = 1 - \epsilon, \\
   Pr[y^\epsilon = \tilde{l}_i] = \frac{\sum_{j=1}^{L} \theta_j x_j^p}{L}, \forall i \in \{1, \ldots, L\}, \\
   Pr[y^\epsilon = 0] = 1 - Pr[y^\epsilon = \bar{x}^p] - \sum_{i=1}^{L} Pr[y^\epsilon = \tilde{l}_i].
   \end{array} \right.
   \]

Why sampling?
To Sample

Why to Sample

Optimal value of Perturbed Problem

\[ \text{POPT} = (P\vec{b})^T \vec{x}^p \geq (1 - \epsilon)OPT \]

Perturbation Matrix

Optimal value of the original Allocation Problem

So we cannot directly use \( \vec{x}^p \) as the Allocation Solution to the original Allocation Problem. We need to find another feasible solution \( \vec{y}^\epsilon \) such that

\[
E[\vec{b}^T \vec{y}^\epsilon] = \vec{b}^T (P^T \vec{x}^p) \geq (1 - \epsilon)OPT
\]
Sketch of the Allocation Algorithm

1: Perturb bidding prices in the allocation problem
   \[
   \overline{b}_i = (1 - \epsilon) b_i + \frac{\theta_i \sum_{j=1}^{L} b_j}{L}, \forall i \in [L].
   \]

   How to perturb?

2: Exactly solve the perturbed problem to obtain its optimal solution \( x^p \)
   How to exactly solve it? Pareto Optimal-based DP

3: Sample allocation solution \( y^\epsilon \) following a distribution:
   \[
   \Omega(\overline{x}^p) = \left\{
   \begin{array}{ll}
   Pr[\overline{y}^e = \overline{x}^p] = 1 - \epsilon, \\
   Pr[\overline{y}^e = \overline{l}_i] = \frac{\sum_{j=1}^{L} \theta_j x_j^p}{L}, \forall i \in \{1, ..., L\}, \\
   Pr[\overline{y}^e = \overline{0}] = 1 - Pr[\overline{y}^e = \overline{x}^p] - \sum_{i=1}^{L} Pr[\overline{y}^e = \overline{l}_i].
   \end{array}
   \right.
   \]

   Why sampling?
   \[
   E[\overline{b}^T \overline{y}^e] = \overline{b}^T (\overline{P}^T \overline{x}^p) \geq (1 - \epsilon) OPT
   \]
How to Achieve Polynomial Expected Running Time?

Smoothed Analysis can prove that our Exact Algorithm has an expected polynomial running time in solving the randomized perturbed problem.

High-level idea:
Adversary chooses $\vec{b}$
We slightly perturb it at random
Worst-case instances are ruled out
1. Calculate $\bar{y}^c$

2. Payment rule: opportunity cost according to

$$p_i(\bar{y}^c) = \bar{b}_i^T \bar{y}_i^c - (\bar{b}^T \bar{y}^c - b_i y_i^c), \forall i \in [L].$$
How to Guarantee Truthfulness?

- It has been proved that Maximal-in-distributional Range (MIDR) Mechanism combined with Randomized VCG payment yields a truthful mechanism.

- Our Allocation Algorithm can be proved to be an MIDR.
Each job in Google Cluster Data contains multiple tasks, with information on resource demands (CPU, RAM, Disk). We translate each job into a bundle bid, and each task in the job into a VM in the bundle bid.

- **Default Parameters:**
  - $\epsilon=0.05$, # of users = 500, # of data centers = 8,
  - average # of bids each user submits = 4,
  - capacity of each type of resource in each data center = (overall amount of this resource required in this data center in all the submitted bid bundles) $\times$ (a random factor)
Performance Evaluation

![Graph showing approximation ratio vs. number of users for different values of D (D=4, D=8, D=12)].
Performance Evaluation
Performance Evaluation

Compared with Zhang et. al in their INFOCOM’14 paper
Summary

1. Polynomial expected running time and $(1-\varepsilon)$-approximation ratio are achieved by perturbation-based randomized algorithm for resource allocation (MIDR)

2. Truthfulness is guaranteed by MIDR allocation rule + randomized VCG payment

2. Performance is evaluated by trace-driven simulations
Thank you!

Q&A