A Distributed Platform for Mechanism Design

Krzysztof R. Apt
joint work with
Farhad Arbab and Huiye Ma
acknowledgments to
Kees Blom and Han Noot

CWI and University of Amsterdam
Executive Summary

We describe design of a structured, highly flexible platform for distributed mechanism design.

The system is built as a sequence of layers.

Lower layers deal with the operations relevant for distributed computing.

Upper layers deal with the relevant aspects of the mechanism design.

Specific applications are realized as instances of the top layer.

The implementation supports various instances of (sequential) Groves mechanisms, and Walker mechanism.
Reminder: Decision problems

Assume

- players $1, \ldots, n$,
- set of decisions $D$,
- for each player a set of types $\Theta_i$ and a utility function $v_i : D \times \Theta_i \rightarrow \mathbb{R}$

that he wants to maximize.

- Decision rule: a function $f : \Theta_1 \times \cdots \times \Theta_n \rightarrow D$.

We call

$$(D, \Theta_1, \ldots, \Theta_n, v_1, \ldots, v_n, f)$$

a decision problem.
Tax-based Mechanisms: Classical View

One studies the following sequence of events:

(i) each player $i$ receives a type $\theta_i$,

(ii) each player $i$ announces to the central authority a type $\theta'_i$; this yields a joint type $\theta' := (\theta'_1, \ldots, \theta'_n)$,

(iii) the central authority computes

- decision $d := f(\theta')$,
- the sequence of taxes $t := g(\theta')$, where $g : \Theta \to \mathbb{R}^n$,

and

- communicates to each player $i$ decision $d$ and the tax $|t_i|$ he needs to pay to/receive from the central authority.

(iv) the resulting utility for player $i$: $u_i(d, t) := v_i(d, \theta_i) + t_i$. 
Decision rules

A decision rule $f$ is called

- **efficient** if for all $\theta \in \Theta$ and $d' \in D$

\[
\sum_{i=1}^{n} v_i(f(\theta), \theta_i) \geq \sum_{i=1}^{n} v_i(d', \theta_i),
\]

- **strategy-proof** if for all $\theta \in \Theta$, $i \in \{1, \ldots, n\}$ and $\theta'_i \in \Theta$

\[
v_i(f(\theta_i, \theta_{-i}), \theta_i) \geq v_i(f(\theta'_i, \theta_{-i}), \theta_i).
\]
Specific Tax-based Mechanisms

- Groves mechanisms: by specific use of taxes (function $g$) truth-telling (reporting $\theta'_i := \theta_i$) becomes a dominant strategy.

- Groves Theorem Suppose $f$ is efficient. Then in each Groves mechanism the decision rule $(f, g)$ is strategy-proof w.r.t. resulting utility functions $u_1, \ldots, u_n$.

- Tax-based mechanism is
  - balanced if $\sum_{i=1}^{n} t_i(\theta) = 0$ for all $\theta$,
  - feasible if $\sum_{i=1}^{n} t_i(\theta) \leq 0$ for all $\theta$.

- VCG mechanism: an instance of Groves mechanism that is feasible.
Distributed Mechanisms

- Originally proposed in (Parkes and Schneidman ’04) for VCG mechanisms.

- First distributed implementation: (Petcu, Faltings and Parkes ’06) for VCG mechanisms and VCG mechanisms with redistribution.

- Both papers require a bank process that is in charge of the computation of taxes.
Our Approach

The only central authority is tax authority; it is needed only for unbalanced mechanisms, players interested in ‘the game’ need to register, this allows us to deal with unknown number of players, the players whose registration is accepted inform other registered players about their types, once a registered player learns that he has received the types from all registered players:
  he computes the decision and the taxes, sends this information to other registered players, and terminates his computation.
Design Decisions

- The system is built as a sequence of layers.
- **Lower layers:**
  - support generation of globally unique process identifiers,
  - support generic broadcast command,
  - its implementation assumes only that each pair of players is connected by a path of neighbours,
  - this allows us to deal with arbitrary network topologies,
  - provide distributed termination detection algorithm,
  - it allows us to separate and to synchronize various phases of distributed computation (barrier synchronization).
Design Decisions (2)

Upper layers:

- support registration process,
- support communication between player processes (and tax authority),
- each tax computation yields identical scheme ‘who pays how much to whom’,
- remaining deficit (if any) is sent to tax authority.
Fault-tolerance and Security

- Low-level asynchronous send supports fault-tolerance at the distributed systems level.
- Redundancy in player processes computation supports fault-tolerance at the mechanism level: decision can be taken and taxes computed even after some player processes crash,
- Interaction between players (users) and the system is reduced to a minimal GUI. This ensures that players cannot tamper with the system.
- Use of locks in player process computations control the flow of information.
Architecture

Player GUI

Player Process

Tax Authority Process

Tax Authority S.I.

High Level Communication

Registry

BTTF

Low Level Communication
Details

Lower layers (9K lines of Java code (Kees Blom)):

- **Low Level Communication**: supports generation of unique process identifiers and asynchronous targeted communication between processes.
- **BTTF**: implements fault-tolerant distributed termination detection (DTD) algorithm.

Software for message passing between internet-based parallel processes (Han Noot).
Details (2)

- Upper layers (3.5K lines of Java code (Huiye Ma)):
  - **High Level Communication and Registry**: supports indirect communication between the processes in a distributed system, supports registration in local registries.
  - **Tax Authority Software Interface**: supports registration of tax authority process in local registry.
  - **Tax Authority Process**: only used for non-balanced mechanisms; to collect the deficit.
  - **Player Process**: used to implement details of specific mechanisms.
(Simplified Version, Balanced Mechanisms)

- process $p_i$ representing player $i$ is created,
- $p_i$ obtains player $i$’s type,
- $p_i$ signs in at the local registry,
- all messages sent to $p_i$ are locked and stored,
- if $p_i$’s registration is confirmed, it broadcasts $i$’s type (and otherwise it terminates),
- the lock of $p_i$ is open,
- $p_i$ invokes the DTD algorithm. When it ends $p_i$ has received all the types,
- $p_i$ computes decision and tax schemes of the registered players and broadcasts them,
- $p_i$ invokes the DTD algorithm and terminates.
Implemented Examples

- Vickrey auction,
- Vickrey auction with redistribution (Cavallo '06),
- Public project problems,
- Unit demand auction (uses Kuhn-Munkres algorithm to compute maximum weighted matching),
- Single minded auction (uses a dynamic algorithm developed by V. Markakis),
- Sequential Groves mechanisms (Apt and Estévez-Fernández '07)
- Walker mechanism (Walker '81): implementation in Nash equilibrium.
Conclusions

- Platform for \textit{distributed mechanism design} realized as a sequence of \textit{layers}.
- This allows us to \textit{separate} distributed computing aspects from mechanism design matters,
- Intentionally limited GUI limits the interaction between the players and the system to a minimum. This ensures security.
- \textbf{Fault-tolerance} is supported on the distributed computing and on the mechanism level.
- \textbf{Flexible design}: for each application \textit{only} top layer needs to be instantiated.
- Platform can be also used for \textit{repeated} mechanisms (e.g., continuous double auctions).
That’s not all: A Small Demo . . .

Single Minded Auction:

- 5 players,
- 2 local registries,
- tax authority,
- 3 items for sale,
- players bids: A: 20:(1,2), B: 50:(3), C: 32:(2), D: 60:(2,3), E: 19:(1).
- generated allocation: (3:B, 28), (2:C, 10), (1:E, 0).

The allocation is computed using a dynamic programming algorithm (V. Markakis).
Player A's Interface

Single Minded Auction

input your type:

Information board:

register

submit
A Distributed Platform for Mechanism Design

Player B's Interface

Single Minded Auction

input your type: X

Information board:
Single Minded Auction

input your type:

Information board:

-Phase1: The player has registered successfully.
A Distributed Platform for Mechanism Design

Player B's Interface

Single Minded Auction

Register

Input your type:

Submit

Information board:

Phase 1: The player has registered successfully.
A Distributed Platform for Mechanism Design

Player A's Interface

Single Minded Auction

input your type: 

20:(1,2)

Information board:

- Phase1: The player has registered successfully.
- Phase2: The player has submitted his type: 20:(1,2)
Player B's Interface

Single Minded Auction

input your type: 50:(3)

Information board:

-Phase1: The player has registered successfully.
-Phase2: The player has submitted his type: 50:(3)
- Phase 1: The player has registered successfully.
- Phase 2: The player has submitted his type: 20:(1,2)
- Phase 3: The player has computed the tax scheme (3,−1,28),(4,−1,10)) and multicast to others.
- Phase 4: The player has received the information about the revenue 38 from the tax authority.
- Phase 4: This ends the game.
**Player B's Interface**

- **Single Minded Auction**
- **Register**

**Input your type:**

50:(3)

**Submit**

---

**Information board:**

- Phase 1: The player has registered successfully.
- Phase 2: The player has submitted his type: 50:(3)
- Phase 3: The player has received the tax scheme ((3, -1, 28), (4, -1, 10)) multicast by others.
- Phase 3: The player has submitted tax 28 to the tax authority.
- Phase 4: The player has received the information about the revenue 38 from the tax authority.
- Phase 4: This ends the game.
A Distributed Platform for Mechanism Design