An Asynchronous Distributed Algorithm for Solving Stochastic Unit Commitment

Ignacio Aravena and Anthony Papavasiliou

Solving the sustainability puzzle

ENGIE Chair research problems and methodology

Integrated transmission/distribution operation

Renewable integration

Parallel computing

Demand response

Resource planning

Market design

Stochastic programming

ColorPower: residential demand response business models based on quality differentiated service

Rewarding flexible capacity in the Belgian electricity market
1. Motivation
2. Preliminaries
3. Asynchronous distributed block-coordinate subgradient method for dual minimization
4. Primal recovery
5. High performance computing implementation
6. Numerical results
   - Western Electricity Coordinating Council (WECC) system
   - Central Western European (CWE) system
7. Conclusions
Motivation
Renewables making headlines

Germany: Nuclear power plants to close by 2022

Denmark aims for 100 percent renewable energy in 2050

California to nearly double wind, solar energy output by 2020 -regulator
Sequential electricity markets

Day-ahead market model is a large-scale mixed-integer linear program, known as the **unit commitment problem**
Stochastic unit commitment problem

- Stochastic unit commitment models uncertainty endogenously in power system operations

Uncertainty: renewable supply, contingencies

Recourse: generator dispatch, demand response, transmission control, storage
Motivation of the present work

- Stochastic unit commitment offers advantages over deterministic reserve policies for coping with uncertainty (Takriti and Birge, 1996), (Papavasiliou and Oren, 2013)

- Stochastic unit commitment has failed to become an industry standard:
  - Market design compatible with treatment of uncertainty
  - Difficulty and scale of stochastic unit commitment models

- Decomposition and parallelization have shown promise to solve stochastic unit commitment models (Cheung et al., 2015), (Kim and Zavala, 2015)
The two-stage stochastic unit commitment problem can be formulated as

\[
\max_{u,v,w} \sum_{i=1}^{N} (c_i^T v_i + d_i^T w_i)
\]

s.t. \( v_i - u = 0, \quad i = 1, \ldots, N \)

\( (v_i, w_i) \in D_i, \quad i = 1, \ldots, N \)

\( u \in \mathcal{U} \)

- \( u \) corresponds to non-anticipative commitment and production variables of generators
- \( v_i \) are local copies of \( u \) for each scenario \( i \)
- \( w_i \) are commitment, production and transmission recourse variables

Scenario decomposition: relax non-anticipativity constraints in order to compute a solution iteratively
Parallelization of common decomposition methods leads to synchronous parallel algorithms, e.g. (Papavasiliou and Oren, 2013), (Cheung et al., 2015)

Execution of a synchronous parallel decomposition method.
Load profile of a synchronous parallel decomposition method. Red area indicates idle processors.

- Evaluation times of subproblems (SP\textsubscript{i}) can vary significantly:
  - Up to 48x across subproblems within the same iteration
  - Up to 25x across iterations for the same subproblem
Preliminaries
Block-coordinate descent method

☐ Each iteration performs a line search on a subset of variables (Tseng and Yun, 2009), (Fercoq and Richtárik, 2013), (Wright, 2015)

\[
\min_{x \in \mathbb{R}^N} f(x_1, x_2, \ldots, x_n)
\]

1. Let \( k := 0 \), \( x^k := \bar{x}^0 \)

2. Select block-coordinate \( j(k) \) (cyclically, at random)

3. Compute \( t^k := \arg \min_t f(x_1^k, \ldots, x_{j(k)-1}^k, t, x_{j(k)+1}^k, \ldots, x_n^k) \)

4. Let \( x_j^{k+1}(k) := t^k \), \( x_i^{k+1} := x_i^k \ \forall i \neq j(k) \)

5. Let \( k := k + 1 \) and return to 2.
Block-coordinate descent method

The function $f(x, y) = 5x^2 - 6xy + 5y^2$ is plotted in the diagram.
The stochastic unit commitment problem

\[
\max_{u, v, w} \sum_{i=1}^{N} (c_i^T v_i + d_i^T w_i)
\]

s.t. \( v_i - u = 0, \quad i = 1, \ldots, N \) \( (x_i) \)

\( (v_i, w_i) \in D_i, \quad i = 1, \ldots, N \)

\( u \in U \)

- \( U \) is a bounded convex set, described by linear constraints
- \( D_i \) is a bounded non-convex set, described by mixed integer-linear constraints
- \( x_i \) are dual multipliers associated to non-anticipativity constraints
Scenario decomposition

- Lagrange relaxation of non-anticipativity constraints leads to the following separable dual problem

\[
\min_{x \in \mathbb{R}^m} f_0(x) + \sum_{i=1}^N f_i(x_i)
\]

where \( f_0 \) and \( f_i \) are defined according to

\[
f_0(x) := \max_{u \in \mathcal{U}} \left( - \sum_{i=1}^N x_i^T u \right) u
\]

\[
f_i(x_i) := \max_{(v,w) \in \mathcal{D}_i} \left( (c_i^T + x_i^T)v + d_i^T w \right) \quad i = 1, \ldots, N
\]

- \( f_0 \) and \( f_i \) are non-differentiable convex functions

- Evaluation of \( f_0 \) requires to solve an LP, while evaluation of \( f_i \) involves solving a large-scale MILP
Solution strategy for SUC

- Each evaluation of the dual function gives **upper bounds** to the SUC problem
- \( v_i \in \partial f_i(x_i), \quad i = 1, \ldots, N \) are (typically) primal feasible first-stage solutions for which we can evaluate their second-stage cost
- Primal feasibility recovery gives us a **lower bound** to the SUC problem
- Lower and upper bounds are used for deciding on termination
Smooth approximation of $f_0$

- In order to obtain convergence guarantees, following (Nesterov, 2005), (Fercoq and Richtárik, 2013), we replace the non-decomposable part of the objective $f_0$ by the following smooth approximation

$$f_0^\mu(x) := \max_{u \in \mathcal{U}} \left( \left( - \sum_{i=1}^{N} x_i^T \right) u - \frac{1}{2} \mu \| u - u_0 \|_2^2 \right)$$

- $f_0^\mu$ is a differentiable function with Lipchitz gradient with constant $L_0^\mu$

- We focus then on minimizing the following approximation of the dual problem of stochastic unit commitment

$$\min_{x \in X} f(x) = f_0^\mu(x) + \sum_{i=1}^{N} f_i(x_i)$$
Contributions of the present work

- We use scenario decomposition and propose an asynchronous block-coordinate subgradient method for minimizing the Lagrangian dual of stochastic unit commitment (convex, non-differentiable).

The proposed method does not perform a line search along coordinates on each iteration (Tseng and Yun, 2009), (Fercoq and Richtárik, 2013).

- We propose primal recovery heuristics

- We implement the proposed asynchronous algorithm on a high performance computing cluster. The algorithm is able to solve industry-scale instances within the same time frame of deterministic unit commitment.
Asynchronous distributed block-coordinate subgradient method
Consider the randomized coordinate descent method (Nesterov, 2012):

1. Let $k := 0$, $x^k := x^0$

2. Select component $j(k)$ uniformly at random from $\{1, \ldots, N\}$

3. Compute $\nabla f^\mu_0(x^k)$ and $g(j(k), x^k_{j(k)}) \in \partial f(j(k))(x^k_{j(k)})$

4. Perform update according to

$$x^{k+1} := \mathcal{P}_X \left[ x^k - \lambda_k \cdot I^T_{j(k)} \left( I_{j(k)} \nabla f^\mu_0(x^k) + g(j(k), x^k_{j(k)}) \right) \right]$$

5. Let $k := k + 1$ and return to 2.
Convergence of the serial method

☐ Consider the expected update direction of the serial method,

$$\mathbb{E}[I_J^T (I_J \nabla f_0^\mu (x^k) + g(J, x^k_J)) | x^k],$$

where $J$ is a discrete uniformly distributed variable on $\{1, \ldots, N\}$.

☐ The expected update direction coincides with the direction of a subgradient of $f$ at $x^k$:

- This property requires $f_0^\mu$ to be smooth
- The serial method is a **stochastic subgradient method** (Ermoliev, 1983)

☐ Provided we choose a diminishing, non-summable and square summable stepsize $\lambda_k$, the algorithm will converge to an optimal solution with probability 1.
Asynchronous method: computation model

The **Subgradient computation system** evaluates $\nabla f^\mu_0$ and subgradients of component functions in parallel.

The **Updating system** stores the last gradient evaluated $\nabla f^\mu_0(x^{k-l(k)})$ and the latest subgradients evaluated $g(i, x^{k-\ell(i,k)}) \forall i$.

Updates are performed using the lastly available information after each evaluation of a subgradient.

Delays $l(k)$ and $\ell(i, k)$ appear because evaluation of gradients and subgradients is not instantaneous.

Conceptual computation model (Nedić et al., 2001).
The *Updating system* performs the following operations:

1. Wait for next subgradient evaluation. Store new information when received.

2. Select component \( j(k) \) uniformly at random from \( \{1, \ldots, N\} \)

3. Perform update according to

\[
\mathbf{x}^{k+1} := \mathcal{P}_X \left[ \mathbf{x}^k - \lambda_k \cdot I_{j(k)}^T \left( I_{j(k)} \nabla f_{\mu} \left( \mathbf{x}^{k-l(k)} \right) + g\left(j(k), \mathbf{x}^{k-l(j(k),k)} \right) \right) \right]
\]

4. Let \( k := k + 1 \) and return to 1.
Start

\[ f_1(x_1^0) \quad f_2(x_2^0) \quad f_3(x_3^0) \quad f_4(x_4^0) \quad f_5(x_5^0) \]

\[ k \quad j(k) \]
Built-in load balancing for subgradient computation

\[ \begin{align*}
\text{Start} & \quad f_1(x_1^0) \quad f_2(x_2^0) \quad f_3(x_3^0) \quad f_4(x_4^0) \quad f_5(x_5^0) \\
& \quad f_4(x_4^1) \\
\end{align*} \]

\[
\begin{array}{c}
k \\
1 \\
4 \\
\end{array}
\]
Built-in load balancing for subgradient computation

\[
\begin{align*}
\text{Start} & \quad f_1(x^0_1) \quad f_2(x^0_2) \quad f_3(x^0_3) \quad f_4(x^0_4) \quad f_5(x^0_5) \\
& \quad f_4(x^1_4) \quad f_3(x^2_3) \\
\frac{k}{j(k)} & \quad \begin{array}{c} 1 \quad 4 \\ 2 \quad 3 \end{array}
\end{align*}
\]
Built-in load balancing for subgradient computation

\[ \begin{align*}
\text{Start} & \quad f_1(x_1^0) \quad f_2(x_2^0) \quad f_3(x_3^0) \quad f_4(x_4^0) \quad f_5(x_5^0) \\
& \quad f_4(x_4^1) \quad f_3(x_3^2) \\
& \quad f_3(x_3^2) \\
& \quad f_2(x_2^3) \\
\end{align*} \]

\[ \begin{array}{c|c}
k & j(k) \\
1 & 4 \\
2 & 3 \\
3 & 2 \\
\end{array} \]
Built-in load balancing for subgradient computation

\[
\begin{align*}
\text{Start} & \quad f_1(x_1^0) \quad f_2(x_2^0) \quad f_3(x_3^0) \quad f_4(x_4^0) \quad f_5(x_5^0) \\
& \quad f_4(x_4^1) \quad f_3(x_3^2) \quad f_3(x_3^4) \\
& \quad f_3(x_3^2) \\
& \quad f_2(x_2^3) \\
& \quad f_2(x_2^3) \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>$k$</th>
<th>$j(k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
Built-in load balancing for subgradient computation

\[
\begin{align*}
\text{Start} & \quad f_1(x_1^0) \quad f_2(x_2^0) \quad f_3(x_3^0) \quad f_4(x_4^0) \quad f_5(x_5^0) \\
f_4(x_4^1) & \quad f_3(x_3^1) \\
f_1(x_1^5) & \quad f_3(x_3^5) \\
\end{align*}
\]

\[
\begin{array}{c|c}
 k & j(k) \\
\hline
 1 & 4 \\
 2 & 3 \\
 3 & 2 \\
 4 & 3 \\
 5 & 1 \\
\end{array}
\]
Built-in load balancing for subgradient computation

\[
\begin{align*}
\mathbf{f}_1(x^0_1) & \quad \mathbf{f}_2(x^0_2) & \quad \mathbf{f}_3(x^0_3) & \quad \mathbf{f}_4(x^0_4) & \quad \mathbf{f}_5(x^0_5) \\
\mathbf{f}_4(x^1_4) & \quad \mathbf{f}_3(x^2_3) & & \quad \mathbf{f}_3(x^4_3) & \quad \mathbf{f}_3(x^6_3) \\
\mathbf{f}_1(x^5_1) & \quad & & & \\
\end{align*}
\]

\[\begin{array}{cc}
k & j(k) \\
1 & 4 \\
2 & 3 \\
3 & 2 \\
4 & 3 \\
5 & 1 \\
6 & 3 \\
\end{array}\]
Built-in load balancing for subgradient computation

\[
\begin{align*}
&f_1(x_1^0) & f_2(x_2^0) & f_3(x_3^0) & f_4(x_4^0) & f_5(x_5^0) \\
&f_4(x_4^1) & f_3(x_3^2) & f_3(x_3^4) & f_3(x_3^6) & f_2(x_2^7) \\
&f_1(x_1^5) & f_3(x_3^3) & f_3(x_3^3) & f_3(x_3^3) & f_2(x_2^3)
\end{align*}
\]

\[\begin{array}{c|c}
k & j(k) \\
1 & 4 \\
2 & 3 \\
3 & 2 \\
4 & 3 \\
5 & 1 \\
6 & 3 \\
7 & 2 \end{array}\]
Built-in load balancing for subgradient computation

\[
\begin{align*}
\text{Start} & \\
& f_1(x_1^0) & f_2(x_2^0) & f_3(x_3^0) & f_4(x_4^0) & f_5(x_5^0) \\
& f_4(x_4^1) & f_3(x_3^2) & f_3(x_3^4) & f_3(x_3^6) & f_2(x_2^3) \text{ or } f_2(x_2^7) \\
& f_1(x_1^5) & f_4(x_4^8) & f_3(x_3^5) & f_3(x_3^3) & f_2(x_2^1) \\
\end{align*}
\]

\[k \quad j(k)\]
\[
\begin{array}{cc}
1 & 4 \\
2 & 3 \\
3 & 2 \\
4 & 3 \\
5 & 1 \\
6 & 3 \\
7 & 2 \\
8 & 4 \\
\end{array}
\]
Built-in load balancing for subgradient computation

\[
\begin{align*}
\text{Start} & \quad f_1(x_1^0) \quad f_2(x_2^0) \quad f_3(x_3^0) \quad f_4(x_4^0) \quad f_5(x_5^0) \\
& \quad f_4(x_4^1) \quad f_3(x_3^2) \quad f_3(x_3^4) \quad f_3(x_3^6) \\
& \quad f_1(x_1^5) \quad f_3(x_3^3) \quad f_2(x_2^3) \\
& \quad f_1(x_1^9) \quad f_4(x_4^8) \quad f_2(x_2^7) \\
& \quad f_4(x_4^8)
\end{align*}
\]

<table>
<thead>
<tr>
<th>$k$</th>
<th>$j(k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>
Built-in load balancing for subgradient computation

Start

\[
\begin{align*}
&f_1(x_1^0) \\
&f_4(x_4^1) \\
&f_1(x_1^5) \\
&f_1(x_1^9) \\
&f_2(x_2^0) \\
&f_3(x_3^2) \\
&f_3(x_3^4) \\
&f_3(x_3^8) \\
&f_3(x_3^{10}) \\
&f_3(x_3^6) \\
&f_4(x_4^0) \\
&f_5(x_5^0) \\
&f_2(x_2^3) \\
&f_2(x_2^7) \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>k</th>
<th>j(k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
Assumptions

1. **Subgradient boundedness**

There exist $C$ and $D$ such that

$$\sup_{j \in \{1, \ldots, N\}} \left\| I_j \nabla f_0^\mu(x) + g(j, y_j) \right\|_2 \leq C, \quad \sup_{j \in \{1, \ldots, N\}} \left\| g(j, x_j) \right\|_2 \leq D.$$

2. **Delay boundedness**

There exists $L$ such that $l(k) \leq L$ and $\ell(i, k) \leq L$, $\forall i, k$.

3. **Diminishing-bounded stepsize**

$\lambda_k$ is independent from $j(k)$ and there exist positive constants $\tilde{G}$ and $\hat{G}$, such that

$$\tilde{G} \gamma_k \leq \lambda_k \leq \hat{G} \gamma_k, \quad \gamma_k = \frac{1}{(1 + r_k^q)} \quad \forall k, \quad \sum_{k=0}^{\infty} \gamma_k = \infty, \quad \sum_{k=0}^{\infty} \gamma_k^2 < \infty.$$
Convergence of the asynchronous method

- Under assumptions 1 and 2, we can show that the expected update direction of the asynchronous method is an approximate subgradient of $f$ at $x^k$.

- In particular, the expected update direction

$$\mathbb{E}[I_J^T (I_J \nabla f_0^\mu (x^{k-l(k)}) + g(J, x^{k-\ell(j,k)})) | F_k]$$

where $J$ is a discrete uniform random variable on $\{1, \ldots, N\}$ and $F_k = \{x^k, x^{k-1}, \ldots, x^0\}$, complies with

$$N \cdot (x^k - y)^T \mathbb{E} \left[ I_J^T \left( I_J \nabla f_0^\mu (x^{k-l(k)}) + g(J, x^{k-\ell(J,k)}) \right) \right| F_k \right] \geq f(x^k) - f(y) - \left( C^2 L_0^\mu \sum_{m=k-L}^{k-1} \lambda_m^2 + 2CDN \sum_{m=k-L}^{k-1} \lambda_m \right)$$

- The asynchronous method can be shown to converge to an optimal solution of the dual problem with probability 1, see (Ermoliev, 1983) and (Nedić, 2002).
Unlike most asynchronous algorithm applications, we require computing bounds on the dual objective \( \rightarrow \textbf{duality gap} \)

Recall we are trying to solve the following problem,

\[
\min_{x \in \mathbb{R}^m} f_0(x) + \sum_{i=1}^{N} f_i(x_i)
\]

We can compute an upper bound, after each subgradient evaluation, at the cost of evaluating \( f_0 \) for a composite of the evaluated \( x_i \)'s as follows

\[
UB^k := f_0(\left[ (x_1^{k-\ell(1,k)})^T \ldots (x_N^{k-\ell(N,k)})^T \right]^T) + \sum_{j=1}^{N} f_j(x_j^{k-\ell(j,k)})
\]
Primal recovery
We consider 4 methods for generating primal candidates:

1. First-in-first-out (FIFO)
   Evaluating solutions to scenario subproblems as they arrive

2. Random (RND)
   Selecting solutions to scenario subproblems at random

3. Last-in-first-out (LIFO)
   Evaluating solutions to scenario subproblems in reverse order

4. Importance sampling recombination heuristic (IS)
   Recall that \( f_i(x_i) := \sup_{(v,w) \in D_i} \left( (c_i^T + x_i^T)v + d_i^T w \right) \) and let \( \bar{v}_i^* \) be the last solution to scenario subproblem \( i, \forall i \)

   (a) Pick a sample from \( \{1, \ldots, N\} \) based on the estimated importance of each scenario.

   (b) Average the \( \bar{v}_i^* \)'s of the sample and project the result onto the feasible set of \( v \).
High performance computing implementation
Evaluating $f_0^\mu, f_0 ...$

Slave $s_1$

Evaluating $f_{j(k)} ...$

Slave $s_2$

$\sum_{i=1}^{N} x_i^k, \bar{u}_{\text{best}}, \sum_{i=1}^{N} f_i(x_i^\ell)$

$\nabla f_0^\mu(x^k), f(x^\ell)$

$\bar{u}, i$

Slave $s_3$

Projecting onto $\mathcal{V} ...$

$\tilde{v}$

$\bar{v}$

Slave $s_4$

Evaluating 2nd stage ...
Slave operations

1. Wait for next **task** from Master.
2. Execute **task**.
3. Send **result** to Master.
4. Return to 1.
Primal projection

Slave $s_1$
Evaluating $f_0^\mu, f_0$ ...

$\sum_{i=1}^{N} x_i^k, \bar{u}_{\text{best}}, \sum_{i=1}^{N} x_i^\ell$,

$\nabla f_0^\mu(x^k), f(x^\ell)$

Master

Slave $s_2$
Evaluating $f_{j(k)}$ ...

$x_{j(k)}^k, g(j(k), x_{j(k)}^k), f_{j(k)}(x_{j(k)}^k)$

$\bar{u}_{t, i}$

Slave $s_3$
Projecting onto $\mathcal{V}$ ...

$\bar{v}$

$\bar{u}$

Slave $s_4$
Evaluating 2nd stage ...

$\sum_{i=1}^{N} x_i^k, \bar{u}, \sum_{i=1}^{N} x_i^\ell$,

$\nabla f_0^\mu(x^k), f(x^\ell)$

$\bar{v}$

$\bar{u}$

$\bar{u}'_{t, i}$

$\bar{v}$

$\bar{u}$

$\bar{u}'_{t, i}$

$\bar{v}$

$\bar{u}$

$\bar{u}'_{t, i}$

$\bar{v}$

$\bar{u}$

$\bar{u}'_{t, i}$

$\bar{v}$

$\bar{u}$
Second stage subproblem

Slave $s_1$
Evaluating $f_0^\mu, f_0$ ...

Slave $s_2$
Evaluating $f_{j(k)}$ ...

Master

Slave $s_3$
Projecting onto $\mathcal{V}$ ...

Slave $s_4$
Evaluating 2nd stage ...

$$\sum_{i=1}^{N} x_{i}^{k}, \bar{u}_{\text{best}}, \sum_{i=1}^{N} x_{i}^{\ell}, \nabla f_0^\mu(x^k), f(x^\ell)$$

$$\sum_{i=1}^{N} f_{i}(x_{i}^{\ell}), \sum_{i=1}^{N} x_{i}^{\ell}$$

$$x_{i}^{k}, x_{j(k)}^{k}, g(j(k), x_{k}^{k}), f_{j(k)}(x_{j(k)}^{k})$$

$$\tilde{u}, \tilde{v}$$

$$\bar{u}^t, i, h_i(\bar{u}^t)$$
Dual scenario subproblem

Slave $s_1$
Evaluating $f_0^\mu, f_0$ ...

Slave $s_2$
Evaluating $f_{j(k)}$ ...

Slave $s_3$
Projecting onto $\mathcal{V}$ ...

Slave $s_4$
Evaluating 2nd stage ...

Master

\[ \sum_{i=1}^{N} x_i^k, \tilde{u}_{\text{best}}, \sum_{i=1}^{N} x_i^\ell, \nabla f_0^\mu(x^k), f(x^\ell) \]

\[ \sum_{i=1}^{N} x_i^k, \tilde{u}, \sum_{i=1}^{N} x_i^\ell \]

\[ g(j(k), x_{j(k)}^k), f_{j(k)}(x_{j(k)}^k) \]

\[ \tilde{v}, \tilde{u}, i \]

\[ h_i(\tilde{u}_i) \]
Non-scenario dual subproblems

Master

Slave $s_1$
Evaluating
$f^\mu_0, f_0$ ...

Slave $s_2$
Evaluating
$f_{j(k)}$ ...

Slave $s_3$
Projecting
onto $V$ ...

Slave $s_4$
Evaluating
2nd stage ...

\begin{align*}
\sum_{i=1}^{N} x^k_i, \bar{u}^{\text{best}}, \sum_{i=1}^{N} f_i(x^\ell_i) \\
\nabla f^\mu_0(x^k), f(x^\ell) \\
g(j(k), x^k_{j(k)}), f_{j(k)}(x^k_{j(k)}) \\
\tilde{v}, i \\
h_i(\bar{u}^t) \\
\end{align*}
Intensively parallel tasks

\[ \sum_{i=1}^{N} x_i^k, \bar{u}_{\text{best}} \]

\[ \sum_{i=1}^{N} f_i(x_i^\ell), \sum_{i=1}^{N} x_i^\ell \]

\[ \nabla f_0^\mu(x_k^i), f(x^\ell) \]

\[ \sum_{i=1}^{N} f^\mu_0(x_k^i), f(x^\ell) \]

\[ \bar{v} \]

\[ u \]

\[ \bar{u}^t, i \]

\[ h_i(\bar{u}^t) \]

\[ \bar{v} \]

\[ u \]

\[ H \]

\[ \tilde{v} \]

\[ \bar{u}^t \]

\[ x_k^j(k) \]

\[ g(j(k), x_k^j(k)), f(j(k), x_k^j(k)) \]

\[ \bar{v} \]

\[ u \]

\[ \bar{u}^t, i \]

\[ h_i(\bar{u}^t) \]

\[ \bar{v} \]

\[ u \]

\[ H \]

\[ \tilde{v} \]

\[ \bar{u}^t \]

\[ x_k^j(k) \]

\[ g(j(k), x_k^j(k)), f(j(k), x_k^j(k)) \]
Master operations

1. Wait for next result from any Slave
2. Receive result from Slave $s$ and post-process it
3. Check bounds: if $UB - LB < \epsilon$ terminate
4. Decide next task
5. If task is dual scenario:
   (a) Select $j$ at random from $\{1, \ldots, N\}$ and update $x_j$
   (b) Send new $x_j$ to Slave $s$
Else, pre-process task accordingly and send it to Slave $s$
6. Return to 1.
Numerical results and conclusions
Numerical results

- The asynchronous algorithm is implemented in C using the Xpress C API and using MPI to handle communications between subprocesses.

- Instances read from SMPS files with explicit periods, preserving lazy constraints.

- Subproblems were solved to 1% optimality.

- Numerical experiments ran on the Cab cluster, hosted at the Lawrence Livermore National Laboratory.

- We ran experiments using the WECC system (Papavasiliou et al., 2015) and the CWE system (Aravena and Papavasiliou, 2017)

- 8 day types per instance: 4 seasons, one weekday and one weekend day per season
## Problem sizes

Sizes per scenario of stochastic unit commitment instances in the literature.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Rows</th>
<th>Columns</th>
<th>Non-zeros</th>
<th>Integers</th>
<th>Subproblem solution time [s], avg. (max.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WECC₁</td>
<td>69 447</td>
<td>28 943</td>
<td>240 724</td>
<td>4 080</td>
<td>9.4 (25.7)</td>
</tr>
<tr>
<td>WECC²</td>
<td>34 441</td>
<td>23 090</td>
<td>139 394</td>
<td>3 074</td>
<td>8.3 (67.9)</td>
</tr>
<tr>
<td>EDF³</td>
<td>812 906</td>
<td>73 562</td>
<td>–</td>
<td>26 122</td>
<td>– (–)</td>
</tr>
<tr>
<td>CWE⁴</td>
<td>609 589</td>
<td>390 075</td>
<td>1 941 270</td>
<td>9 753</td>
<td>3 383.2 (7 851.8)</td>
</tr>
</tbody>
</table>

Largest instances solved in the present study.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Scenarios</th>
<th>Rows</th>
<th>Columns</th>
<th>Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>WECC-182²</td>
<td>1000</td>
<td>36 267 000</td>
<td>23 091 826</td>
<td>3 074 000</td>
</tr>
<tr>
<td>CWE⁴</td>
<td>120</td>
<td>74 755 320</td>
<td>46 822 372</td>
<td>1 170 360</td>
</tr>
</tbody>
</table>

¹(Cheung et al., 2015), ²(Papavasiliou et al., 2015), ³(van Ackooij and Malick, 2016), ⁴(Aravena and Papavasiliou, 2017)
130 thermal generators, 182 nodes, 319 lines, hourly resolution, 24 hour horizon, generation and transmission contingencies, multi-area renewable production
## WECC: Solution times

<table>
<thead>
<tr>
<th>$N$</th>
<th>Step size</th>
<th>Recovery Type</th>
<th># Cores</th>
<th>Dual Share</th>
<th>Solution time [s], avg. (max.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2% optimality</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2% optimality</td>
</tr>
<tr>
<td>10</td>
<td>Dim. 1/k</td>
<td>FIFO</td>
<td>16</td>
<td>0.5</td>
<td>228.1 (792.9)</td>
</tr>
<tr>
<td></td>
<td>Dim. 1/k</td>
<td>RND</td>
<td>16</td>
<td>0.5</td>
<td>229.4 (856.1)</td>
</tr>
<tr>
<td></td>
<td>Dim. 1/k</td>
<td>LIFO</td>
<td>16</td>
<td>0.5</td>
<td>200.6 (739.7)</td>
</tr>
<tr>
<td></td>
<td>Dim. 1/k</td>
<td>IS</td>
<td>16</td>
<td>0.5</td>
<td>178.0 (638.0)</td>
</tr>
<tr>
<td></td>
<td>Polyak</td>
<td>FIFO</td>
<td>16</td>
<td>0.5</td>
<td>148.2 (469.3)</td>
</tr>
<tr>
<td></td>
<td>Polyak</td>
<td>RND</td>
<td>16</td>
<td>0.5</td>
<td>117.8 (392.6)</td>
</tr>
<tr>
<td></td>
<td>Polyak</td>
<td>LIFO</td>
<td>16</td>
<td>0.5</td>
<td>131.2 (446.2)</td>
</tr>
<tr>
<td><strong>Polyak</strong></td>
<td>IS</td>
<td>16</td>
<td>0.5</td>
<td></td>
<td><strong>118.4 (441.4)</strong></td>
</tr>
<tr>
<td>100</td>
<td>Polyak</td>
<td>FIFO</td>
<td>160</td>
<td>0.5</td>
<td>267.6 (325.1)</td>
</tr>
<tr>
<td></td>
<td>Polyak</td>
<td>RND</td>
<td>160</td>
<td>0.5</td>
<td>113.2 (345.6)</td>
</tr>
<tr>
<td></td>
<td>Polyak</td>
<td>LIFO</td>
<td>160</td>
<td>0.5</td>
<td>99.4 (268.3)</td>
</tr>
<tr>
<td><strong>Polyak</strong></td>
<td>IS</td>
<td>160</td>
<td>0.5</td>
<td></td>
<td><strong>95.5 (289.8)</strong></td>
</tr>
<tr>
<td>1000</td>
<td>Polyak</td>
<td>LIFO</td>
<td>256</td>
<td>0.75</td>
<td>723.9 (2 155.2)</td>
</tr>
<tr>
<td><strong>Polyak</strong></td>
<td>IS</td>
<td>256</td>
<td>0.75</td>
<td></td>
<td><strong>411.7 (1 354.5)</strong></td>
</tr>
</tbody>
</table>
Central Western European system

656 thermal generators, 679 nodes, 1073 lines, quarterly resolution, 24 hour horizon, multi-area renewable production
Solution time statistics for WECC instances, over 8 representative day types. All instances use the Polyak stepsize to perform dual updates, the IS primal recovery heuristic and a *DualShare* of 0.75.

<table>
<thead>
<tr>
<th>$N$</th>
<th># Cores</th>
<th>Solution time [s], avg. (max.)</th>
<th>2% optimality</th>
<th>1% optimality</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>96</td>
<td>2580.3 (5908.2)</td>
<td>3806.2 (9279.1)</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>192</td>
<td>2563.7 (5593.3)</td>
<td>3774.2 (8323.4)</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>384</td>
<td>2696.5 (5973.0)</td>
<td>3876.2 (7952.6)</td>
<td></td>
</tr>
</tbody>
</table>
Parallel efficiency plot of the asynchronous algorithm. Plot drawn using WECC, spring weekdays, 100 scenario instance, solved using *Dual Share* 0.75, Polyak stepsize and IS primal recovery.
Estimated idle time of processors when solving SUC using a parallel synchronous algorithm. Average and maximum over 8 day types.

<table>
<thead>
<tr>
<th>System</th>
<th>$N$</th>
<th># Cores</th>
<th>Dual Share</th>
<th>Synchronous idle time [%], avg. (max.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Same conditions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Same conditions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Same conditions</td>
</tr>
<tr>
<td>WECC</td>
<td>10</td>
<td>16</td>
<td>0.5</td>
<td>48.0 (53.2)</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>160</td>
<td>0.5</td>
<td>70.6 (80.4)</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>256</td>
<td>0.75</td>
<td>37.6 (57.0)</td>
</tr>
<tr>
<td>CWE</td>
<td>30</td>
<td>96</td>
<td>0.75</td>
<td>36.4 (47.0)</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>192</td>
<td>0.75</td>
<td>43.6 (62.2)</td>
</tr>
<tr>
<td></td>
<td>120</td>
<td>384</td>
<td>0.75</td>
<td>46.9 (61.2)</td>
</tr>
</tbody>
</table>
Conclusions

- Randomized block-coordinate descent converges without the need for line search
- Synchronous algorithms can lead to idle times of up to 80.4% of the total wall time
- The asynchronous algorithm allows us to solve industrial-scale instances within operationally acceptable time frames
- Future extensions of the present work will focus on:
  - Extensions to multi-stage stochastic unit commitment
  - Integrated optimization of generation, transmission and distribution systems ([Caramanis et al., 2016](#))
Thank you

Contact:

Ignacio Aravena, ignacio.aravena@uclouvain.be
http://sites.google.com/site/iaravenasolis/

Anthony Papavasiliou, anthony.papavasiliou@uclouvain.be
http://perso.uclouvain.be/anthony.papavasiliou/
Appendix
WECC: Dual algorithm progress

Convergence of dual function, winter weekday

\[ 1 - \frac{|UB|}{|LB^*|} \]

- 10 scenarios
- 100 scenarios
- 1000 scenarios

Solution time \( \times (\text{#dual processors/\#scenarios}) \) [secs]
Variation of solution time with the *Dual Share*. Results in the table correspond to WECC, spring weekdays, 100 scenario instance, solved using 8 nodes (96 cores) and a Polyak stepsize.

<table>
<thead>
<tr>
<th>Primal Recovery</th>
<th>Dual Share</th>
<th>Solution time [s], avg. (max.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$k = 0$</td>
<td>$k = 200N$</td>
</tr>
<tr>
<td>IS</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>LIFO</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>IS</td>
<td>0.75</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Incremental method

Each iteration uses gradient information of a part of the objective function (Nedić et al., 2001), (Gürbüzbalaban et al., 2015)

\[
\min_{x \in \mathbb{R}^N} \sum_{i} f_i(x)
\]

1. Let \( k := 0 \), \( x^k := \bar{x}^0 \)
2. Select component \( j(k) \) (cyclically, at random)
3. Compute \( g^k \in \partial f_{j(k)}(x^k) \)
4. Let \( x^{k+1} := x^k - \lambda_k g^k \)
5. Let \( k := k + 1 \) and return to 2.
Incremental method

\[ f(x, y) = 3x^2 - 2xy + 2y^2 + 2x^2 - 4xy + 3y^2 \]
Incremental method

\[ f(x, y) = 3x^2 - 2xy + 2y^2 + 2x^2 - 4xy + 3y^2 \]
Block-coordinate descent method

$$f(x, y) = 5x^2 - 6xy + 5y^2$$
The asynchronous incremental method (Nedić, 2001) could be readily applied to minimize $f$. We lean towards coordinate descent because

- As $x$ approaches an optimal value, the gradient of $f^\mu_0$ will tend to point in the opposite direction to the subgradient of $f_i$. This makes the incremental method susceptible to oscillations.

- Every time the asynchronous incremental method updates $x$ using the gradient of $f^\mu_0$, it will introduce an additional delay on all subgradients currently being computed.

Each block-coordinate update causes an additional delay only on block $j(k)$, if $g(j(k), x_{j(k)}^{k-l'(j(k),k)})$ is being computed.