Implémentation asynchrone d'un algorithme de marché de l'électricité décentralisé

Alyssia Dong, Thomas Baroche, Roman Le Goff Latimier, H. Ben Ahmed

Implémentation asynchrone d'un algorithme de marché de l'électricité décentralisé

Alyssia DONG, Thomas BAROCHE, Roman LE GOFF LATIMIER, Hamid BEN AHMED
Laboratoire SATIE, ENS RENNES

RESUME – Dans un réseau électrique, l'équilibre entre production et consommation d'énergie assure la stabilité du réseau. À travers un marché de l'électricité auquel participent les producteurs et les consommateurs, un consensus est atteint de manière à optimiser les coûts de production tout en respectant la condition d'équilibre des puissances. On s'intéresse dans cet article à la résolution d'un algorithme de marché pair à pair décentralisé. La version asynchrone de cet algorithme est étudiée et comparée à la version synchrone dans un cadre où les échanges de messages se font sur un réseau de communication non idéal, comportant des délais différenciés ainsi que des pertes et retransmissions. On présentera, à travers un cas d'application simple, les paramètres qui permettent une convergence plus rapide de l'algorithme en fonction de l'état du réseau de communication.

Mots-clés – marché pair à pair, résolution asynchrone, ADMM, délais de communication, réseau avec pertes.

1. INTRODUCTION

À mesure que la part des énergies renouvelables augmente dans le mix de production d'électricité, les réseaux électriques deviennent de plus en plus interconnectés et le nombre de producteurs d'électricité augmente significativement. On parle alors de *smart grid*. Le terme rassemble les éléments qui composent le réseau électrique et permettent son bon fonctionnement : on prend en compte en plus des producteurs, consommateurs, des lignes et infrastructures électriques, les éléments de mesures, contrôle et supervision qui permettent de piloter de manière plus fine, plus "intelligente" le réseau. Au réseau électrique se superpose un réseau de communication, comme l'illustre la Fig. 1, qui permet d'effectuer des échanges d'information entre les différents acteurs du smart-grid pour permettre un contrôle décentralisé des producteurs et consommateurs d'électricité. Il apparaît naturel qu'à mesure que le réseau électrique se décentralise, on doive considérer des manières décentralisées de gérer les productions et consommations du réseau.

L'équilibre sur un réseau électrique dépend de l'égalité entre la puissance produite et la puissance consommée à tout instant. Pour assurer cette égalité, il est nécessaire d'anticiper les échanges sur le réseau électrique. Un marché de l'électricité permet de répartir le plan de production à partir des consommations prévues. À l'heure actuelle, la résolution de ce marché se fait de manière centralisée à travers un gestionnaire de marché qui réunit les offres et les demandes pour déterminer un prix de l'électricité, et une répartition de la production et de la consommation sur le réseau.

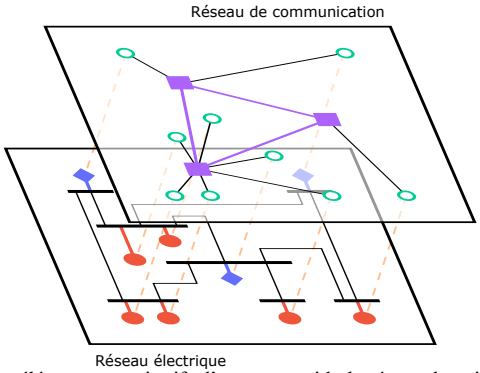

Fig. 1. Deux éléments constitutifs d'un smart grid : le réseau de puissance électrique et le réseau de communication.

Le problème de cette manière de faire est qu'elle centralise la résolution de marché, la rendant de plus en plus complexe à mesure que le nombre d'acteurs augmente. De plus, certains de ces acteurs peuvent être réticents à partager leurs données avec une même entité. Pour pallier ces problèmes, une version décentralisée de résolution de marché est proposée. La décentralisation de la résolution de marché se fait en contrepartie d'une augmentation significative des échanges d'information entre acteurs du marché. Ainsi, le marché est beaucoup plus sensibles aux défauts du réseau de communication. Parmi les différentes façons de résoudre un marché décentralisé, on choisit dans cet article de se concentrer sur un algorithme *pair à pair*, c'est-à-dire où les agents proposent directement de s'échanger de l'énergie, sans passer par un tiers.

Les travaux existants étudient des algorithmes de résolution de marché distribués [1, 2, 3] (les calculs sont distribués mais un agent central subsiste pour coordonner le marché) ou bien complètement décentralisés [4, 5], et proposent notamment des versions asynchrones de résolution de ces algorithmes. L'asynchronisme apporte en théorie une certaine robustesse aux algorithmes de résolution de marché par rapport aux délais de communication variés.

Ces délais de communication découlent soit d'une implémentation réelle d'un réseau de communication, et dans ce cas, ils ne sont pas contrôlés mais seulement subis, ce qui est le cas dans [6, 7], ou bien les délais sont tirés d'une variable aléatoire identiquement et indépendamment distribuée, comme dans [1, 2, 3, 5, 8]. Les modèles de délais de communication proposés restent identiques quels que soient les échanges entre agents, et ne supposent pas que ces délais dépendent de l'emplacement des agents dans un réseau de communication.

On propose dans cet article d'étendre l'algorithme de résolution de marché pair à pair de [9], décentralisé par méthode ADMM [10], à une version asynchrone permettant d'être plus résilient face aux différents défauts et délais du réseau de communication. Cette version permet aussi de proposer des prix différenciés entre acteurs pour pouvoir prioriser certains producteurs par rapport à d'autres (les producteurs renouvelables par exemple). L'étude porte alors sur l'impact du réseau de communication sur les versions synchrone et asynchrone de l'algorithme. Pour ce faire, un modèle de communication simple prenant en compte l'emplacement des agents dans le réseau de communication ainsi que les pertes de message est proposé. Il s'agit alors de comparer les performances des versions synchrones et asynchrones de l'algorithme en fonction de l'état du réseau de communication. On s'assurera que les résultats obtenus sont identiques quelque soit la méthode choisie, et on comparera les temps de convergence.

La section 2 présente l'algorithme de résolution de marché pair à pair. La section 3 introduit le principe d'asynchronisme et en déduit une version asynchrone de l'algorithme, avant de présenter un modèle simple de réseau de communication. L'algorithme est alors implémenté dans la section 4 et les résultats obtenus sont présentés en section 5.

2. RÉSOLUTION DÉCENTRALISÉE DE MARCHÉ

2.1. *Problème à résoudre*

Soit un marché $\Omega = \{1, \dots, N\}$ constitué de N agents indépendants, pouvant être producteurs ou consommateurs. Le problème d'optimisation correspondant au marché est le suivant :

$$\underset{p_i}{\text{minimiser}} \sum_{i\in\Omega} f_i(p_i) \tag{1a}$$

$$telque \quad \sum_{i\in\Omega} p_i = 0 \tag{1b}$$

$$\underline{p_i} \leq p_i \leq \overline{p_i} \quad i \in \Omega \tag{1c}$$

avec

- p_i la puissance échangée par l'agent i, comptée positive si produite par l'agent et négative si consommée,
- $f_i(.)$ le coût de production ou de consommation de l'agent i, fonction quadratique de la puissance p_i,
- $\underline{p_i}$ et $\overline{p_i}$ les limites de production ou consommation de l'agent i.

Il s'agit de minimiser la somme des coûts de tous les agents du marché (1a) de manière à ce que l'équilibre des puissances soit atteint (1b) et que les limites physiques des agents soient respectées (1c). On ne prend pas en compte les contraintes liées aux limitations physiques des lignes du réseau.

2.2. *Distribution du problème*

La complexité du problème (1) augmente avec le nombre N de participants au marché, ce qui rend le passage à l'échelle complexe. Il est donc nécessaire de distribuer le problème pour pouvoir le résoudre lorsqu'il y a beaucoup d'agents. La distribution permet de décomposer le problème (1) en N sous-problèmes beaucoup plus simples, en ajoutant de la communication. De plus, cela procure une certaine confidentialité aux agents ne souhaitant pas partager leurs informations avec une seule entité.

Il existe plusieurs algorithmes distribuant la résolution d'un problème d'optimisation sur plusieurs agents. Notre choix se porte sur un algorithme de résolution de marché pair à pair basé sur [9], qui permet de privilégier certains échanges commerciaux au profit d'autres – préférences hétérogènes – et qui a l'avantage d'être totalement décentralisé : il ne nécessite pas la présence d'un agent central.

Le formalisme choisi pour la résolution décentralisée permet de différencier quelle part de la puissance produite par un agent est consommée par ses partenaires commerciaux. On note ω_i l'ensemble des partenaires commerciaux de l'agent i.

La puissance injectée p_i est alors décomposée en une somme de trades échangés avec les partenaires commerciaux de l'agent concerné :

$$p_i = \sum_{j\in\omega_i} t_{ij} \quad i \in \Omega \tag{2}$$

où ω_i désigne l'ensemble des partenaires commerciaux de l'agent i.

La contrainte d'équilibre du réseau (1b) est équivalente à une contrainte de réciprocité des trades :

$$t_{ij} = -t_{ji} \quad i \in \Omega \quad j \in \omega_i \tag{3}$$

c'est-à-dire que l'agent i s'engage à vendre à l'agent j une quantité égale à ce que l'agent j s'engage à acheter à l'agent i.

Le problème (1) de résolution de marché peut alors se réécrire de la manière suivante :

$$\text{minimiser} \quad \sum_{i\in\Omega} f_i(p_i) \tag{4a}$$

$$\text{tel que} \quad \mathbf{T} = -\mathbf{T}^T \tag{4b}$$

$$p_i = \sum_{j\in\omega_i} t_{ij} \quad i = 1, \dots, N \tag{4c}$$

$$\underline{p_i} \leq p_i \leq \overline{p_i} \quad i \in \Omega \tag{4d}$$

avec $[\mathbf{T}]_{ij} = t_{ij}$ la matrice $(N \times N)$ des trades. Un élément nul t_{ij} de la matrice $\mathbf{T}$ démontre l'absence d'un lien commercial entre les agents i et j, autrement dit : $t_{ij} = 0$ si $j \notin \omega_i$.

2.3. *Décomposition par ADMM*

La décomposition par ADMM permet de diviser le problème global décrit par (4) en N sous problèmes, chacun résolu par un agent. La résolution se fait en plusieurs itérations, dans lesquelles les agents résolvent un problème local et communiquent les résultats intermédiaires à leurs partenaires commerciaux.

Les calculs d'une itération locale d'un agent i sont donnés dans les équations (5).

$$(p_i, \mathbf{t}_i)^{k+1} = \underset{(p_i,\mathbf{t}_i)}{\arg\min}\ f_i(p_i) + \sum_{j\in\omega_i} \frac{\rho}{2}\left(\frac{t_{ij}^k - t_{ji}^k}{2} - t_{ij} + \frac{\lambda_{ij}^k}{\rho}\right)^2 \tag{5a}$$

$$\text{tel que} \quad p_i = \sum_{j\in\omega_i} t_{ij}$$

$$\underline{p_i} \leq p_i \leq \overline{p_i}$$

$$\lambda_{ij}^{k+1} = \lambda_{ij}^k - \rho\,\frac{t_{ij}^{k+1} + t_{ji}^{k+1}}{2} \quad j \in \omega_i \tag{5b}$$

avec

- $\rho > 0$ le facteur de pénalité de l'algorithme,
- $\mathbf{\Lambda} = [\lambda_{ij}]$ la variable duale liée à la contrainte de réciprocité des trades (4b).

L'avancement de l'algorithme est caractérisé par des résidus primal et dual. Ces derniers sont définis comme suit pour chaque agent $i \in \Omega$:

$$r_i^k = \sum_{j\in\omega_i} (t_{ij}^k + t_{ji}^k)^2 \tag{6a}$$

$$s_i^{k+1} = \sum_{j\in\omega_i} (t_{ij}^{k+1} - t_{ij}^k)^2 \tag{6b}$$

Le résidu primal r_i reflète l'équilibre des trades de l'agent i avec ses partenaires commerciaux : il est nécessaire que la quantité d'énergie qu'achète l'agent i à l'agent j corresponde à la quantité d'énergie que vend l'agent j à l'agent i. Ainsi, $r_i^k \xrightarrow[k\to\infty]{} 0$.

Le résidu dual s_i mesure le taux de variation des trades proposés par l'agent i d'une itération à la suivante. Lorsque l'algorithme converge, il n'y a plus de variations : $s_i^k \xrightarrow[k\to\infty]{} 0$.

On introduit aussi les résidus primal et dual globaux en :

$$r^k = \sum_{i\in\Omega} r_i^k \tag{7a}$$

$$s^k = \sum_{i\in\Omega} s_i^k \tag{7b}$$

En général, on considère que l'algorithme a convergé lorsque les résidus primal et dual globaux sont en dessous d'un certain seuil ε.

2.4. *Convexification stricte du problème*

En considérant les équations (4), on s'aperçoit que la quantité à minimiser (4a) est une fonction quadratique des puissances p_i injectées dans le réseau.

Cette fonction est bien strictement convexe lorsque l'on considère seulement les variables $(p_i)_{i\in\Omega}$, mais elle devient non strictement convexe lorsque l'on considère les variables de trades constituant la matrice $\mathbf{T} = [t_{ij}]$. En effet, l'équation (2) montre que pour la solution unique optimale p_i^*, il existe une infinité de solutions $\mathbf{t}_i$.

Un terme est alors ajouté à la fonction objectif (4a) pour convexifier strictement le problème de manière à ce que la solution soit unique en termes de trades :

$$\sum_{i\in\Omega}\left(f_i(p_i) + \gamma \sum_{j\in\omega_i} {t_{ij}}^2\right) \tag{8}$$

Le paramètre γ est appelé *facteur de convexification* et permet d'obtenir la solution qui minimise les échanges t_{ij}^* entre les agents, à puissance échangée avec le réseau p_i^* donnée. Il est réglé de manière à impacter le moins possible le résultat final.

2.5. *Résolution synchrone*

On appelle *résolution synchrone* de l'algorithme la résolution classique où, à chaque itération locale de l'agent i, ce dernier prend en compte les mises à jour de tous ses partenaires commerciaux dans ω_i.

Cela permet le synchronisme de tous les agents du marché, c'est-à-dire qu'à un instant donné, ces derniers se trouvent tous au même numéro d'itération.

Algorithme 1 Algorithme synchrone : procédé local

procédure PROCÉDÉAGENT($\mathbf{t}_i^0, \boldsymbol{\lambda}_i^0$)
  $\mathbf{t}_i \leftarrow \mathbf{t}_i^0$ ▷ Initialisation
  $\boldsymbol{\lambda}_i \leftarrow \boldsymbol{\lambda}_i^0$
  $k \leftarrow 0$
  envoyer t_{ij}^0 à $j \in \omega_i$
  répéter ▷ Répéter jusqu'à convergence
    recevoir t_{ji}^k de $j \in \omega_i$
    si $k \geq 1$ **alors**
      calculer $\boldsymbol{\lambda}_i^k$ avec (5b)
    fin si
    calculer $\mathbf{t}_i^{k+1}$ avec (5a)
    envoyer t_{ij}^{k+1} à $j \in \omega_i$
    $k \leftarrow k + 1$
  tant que résidu local $\leq \epsilon$
fin procédure

Les calculs effectués par un agent i lors d'une itération sont décrits en (5). L'itération est divisée en deux parties :

— La mise à jour en (5b) de la variable duale λ_{ij} représentant le prix marginal de l'échange commercial entre les agents i et j.

— La mise à jour des variables de trades t_{ij} en (5a).

Elle nécessite que l'agent local i ait connaissance des variables de trades associées à ses partenaires commerciaux comme l'illustrent les termes t_{ji} dans (5). Une fois les calculs d'une itération effectués, les résultats mis à jour sont donc communiqués aux agents partenaires. Le procédé associé à un agent local i est décrit dans l'algorithme 1.

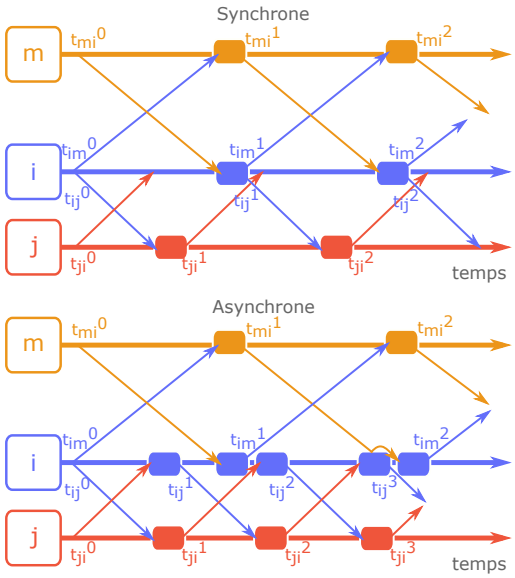

Fig. 2. Schéma temporel des échanges de messages dans un marché à trois agents i, j et k où $\omega_i = \{j, k\}$ et $\omega_j = \omega_k = \{i\}$ dans les cas synchrone et asynchrone. Les flèches représentent les échanges de messages au cours du temps, et les rectangles colorés la durée de calcul d'une itération.

Du fait des aléas de communication, la durée d'une itération est déterminée par le dernier message reçu parmi tous les agents. Si un seul des agents du marché, même s'il n'est partenaire que d'un petit nombre d'agents, est défaillant ou montre un ralentissement conséquent, l'ensemble des agents sont mis en attente lors de la résolution de l'algorithme. Il paraît donc difficile pour cette version synchrone de l'algorithme, bien que décentralisé, de passer à l'échelle au vu des aléas de communication.

3. RÉSOLUTION ASYNCHRONE

3.1. *Algorithme asynchrone*

Pour palier ces problèmes de ralentissement et permettre une implémentation opérationnelle de cette étude, la version asynchrone de l'algorithme est étudiée.

Dans le cas d'une résolution asynchrone, les calculs effectués lors d'une itération locale de l'agent i ne prennent en compte qu'une partie des mises à jour associées aux partenaires commerciaux. En effet l'agent i n'attend la réception que d'un nombre limité de messages avant d'effectuer les calculs pour avancer à l'itération suivante. La Fig. 2 présente les échanges au cours du temps pour un marché à trois agents i, j et m, où l'agent i est le seul partenaire commercial des deux autres agents j et m. On remarque que les échanges entre les agents i et j sont plus rapides que ceux entre i et m.

Considérons le passage de l'itération 1 à l'itération 2 dans le cas synchrone. D'après l'Éq. 5a, l'agent i a besoin de connaître les valeurs t_{ji}^1 et t_{mi}^1 étant donné que les variables t_{ij}^1, t_{im}^1, λ_{ij}^1 et λ_{im}^1 sont connues de l'agent i. Il doit alors attendre l'arrivée des messages provenant de ses deux partenaires commerciaux avant de pouvoir passer à l'itération 2. De leur côté, les agents j et m attendent eux respectivement les informations t_{ij}^1 et t_{im}^1 pour passer à l'itération 2. Or, ces messages ne sont envoyés qu'à l'issue de l'itération 1 de l'agent i, dont le déclenchement dépend de la réception des deux messages précédents t_{ji}^0 et t_{mi}^0. Ainsi, l'avancement global de l'algorithme dépend et est limité par l'agent le plus lent à communiquer l'information à ses partenaires commerciaux.

Considérons maintenant le cas asynchrone, où l'agent i déclenche ses calculs dès la réception d'un message, qu'il provienne de l'agent j ou m. Ainsi, on observe dans la Fig. 2 que les itérations de l'agent j sont bien plus rapides maintenant que son partenaire commercial i n'a pas à attendre le message de l'agent m avant de pouvoir effectuer son calcul et renvoyer le résultat. Étant donné que les échanges entre les agents i et m sont les plus lents, on remarque que l'avancement de l'agent m se fait à la même vitesse que dans le cas synchrone.

On a vu que la résolution asynchrone permet de réaliser un gain de temps sur les itérations. Cependant, le manque d'information nécessite le calcul de plus d'itérations pour atteindre la convergence de l'algorithme. Les informations manquantes sont remplacées par les dernières informations en date.

La notion d'itération globale n'ayant plus lieu d'être, on désigne alors une itération locale à l'agent i : k_i, ainsi qu'une itération propre à l'avancement du trade ij : k_{ij}.

On observe dans les équations (5a) et (5b) l'association des termes t_{ij}^k et t_{ji}^k. Ces termes correspondent aux trades réciproques entre les agents i et j et possèdent le même numéro d'itération k. Dans la version asynchrone de l'algorithme, on s'efforce alors de n'associer que les termes $t_{ij}^{k_{ij}}$ et $t_{ji}^{k_{ji}}$ tels que $k_{ij} = k_{ji}$.

On note $\Phi_i^{k_i} \subset \omega_i$ l'ensemble des partenaires commerciaux dont l'agent i a reçu la variable $t_{ji}^{k_{ji}}$ telle que $k_{ij} = k_{ji}$ durant l'itération locale k_i. Les équations d'une itération de l'agent local i sont données en (9). Elles ne prennent en compte que les données des partenaires commerciaux dont l'agent local i a reçu un message depuis sa dernière itération.

$$\lambda_{ij}^{k_{ij}} = \lambda_{ij}^{k_{ij}-1} - \rho \frac{t_{ij}^{k_{ij}} + t_{ji}^{k_{ji}}}{2} \quad j \in \Phi_i^{k_i} \quad k_{ij} \geq 1 \tag{9a}$$

$$(p_i, \mathbf{t}_i)^* = \underset{(p_i,\mathbf{t}_i)}{\arg\min}\, f_i(p_i) + \sum_{j\in\omega_i} \left[\gamma\, {t_{ij}}^2 + \frac{\rho}{2} \left(\frac{t_{ij}^{k_{ij}} - t_{ji}^{k_{ji}}}{2} - t_{ij} + \frac{\lambda_{ij}^{k_{ij}}}{\rho} \right)^2 \right] \tag{9b}$$

$$\text{tel que} \quad p_i = \sum_{j\in\omega_i} t_{ij}$$

$$\underline{p_i} \leq p_i \leq \overline{p_i}$$

$$t_{ij}^{k_{ij}+1} = t_{ij}^* \qquad j \in \Phi_i^{k_i} \tag{9c}$$

L'algorithme asynchrone qui sera implémenté est décrit dans l'algorithme 2.

Algorithme 2 Algorithme asynchrone : procédé local

procédure PROCÉDÉAGENT($\mathbf{t}_i^0, \boldsymbol{\lambda}_i^0$)
  $\mathbf{t}_i \leftarrow \mathbf{t}_i^0$ ▷ Initialisation
  $\boldsymbol{\lambda}_i \leftarrow \boldsymbol{\lambda}_i^0$
  $k_{ij} \leftarrow 0$ pour $j \in \omega_i$
  envoyer t_{ij}^0 à $j \in \omega_i$
  répéter ▷ Répéter jusqu'à convergence
    recevoir t_{ji}^k de $j \in \Phi_i$
    si $k_{ij} \geq 1$ **alors**
      mettre à jour $\boldsymbol{\lambda}_i$ avec (9a)
    fin si
    mettre à jour $\mathbf{t}_i$ avec (9b-9c)
    envoyer $t_{ij}^{k_{ij}+1}$ à $j \in \Phi_i$
    $k_{ij} \leftarrow k_{ij} + 1$ pour $j \in \Phi_i$
    $k_i \leftarrow k_i + 1$
  tant que résidu local $\leq \epsilon$
fin procédure

On observe que les mises à jour des différentes variables locales de l'agent i ne sont effectuées que si ces variables sont associées aux partenaires $j \in \Phi_i$. Il est alors logique de ne renvoyer les messages contenant les mises à jour de ces variables qu'aux agents éléments de Φ_i à la fin de l'itération locale k_i.

En réalité, Φ_i est déterminé par l'ordre d'arrivée des messages destinés à l'agent i. Le déclenchement de l'itération locale se fait lors de la réception de $N_{mes,i} \leq |\omega_i|$ messages.

Remarque : le cas $N_{mes,i} = |\omega_i|$ est équivalent à l'algorithme synchrone.

3.2. *Modèle de communication*

La détermination du groupe Φ_i à chaque itération locale de l'agent i est déterminée par l'ordre d'arrivée des messages de ses partenaires commerciaux. Un modèle de délai de communication est donc nécessaire pour étudier l'effet de la topologie du réseau de communication sur la convergence de l'algorithme asynchrone. Dans le cadre de cette étude, un modèle très simple est utilisé. Il repose sur le constat qu'un message traversant plus d'éléments de routage du réseau met plus de temps à arriver à destination car il a plus de chance de se retrouver dans une file d'attente.

Soit $L(\Omega) = \{(i,j) \in \Omega^2 \,|\, t_{ij} \neq 0\}$ l'ensemble des liens commerciaux entre les agents du marché Ω. Pour chaque paire (i,j) appartenant à $L(\Omega)$, on définit alors une 'distance' D_{ij} proportionnelle au nombre d'élements sur le chemin entre les agents i et j. On considère que ce chemin demeure inchangé tout au long de la résolution du problème, c'est-à-dire qu'il n'y a pas de protocole de routage dynamique.

Le modèle de délai de communication repose sur une relation affine entre la distance entre deux agents et le délai de communication entre ces deux agents :

$$\Delta T_{ij} = \overline{\Delta T} + \alpha \cdot (D_{ij} - \overline{D}) \tag{10}$$

où $\bar{\cdot}$ représente la moyenne sur toutes les connexions possibles entre agents, donc $\overline{\Delta T}$ est la moyenne des délais et $\overline{D}$ la moyenne des distances, et α désigne le paramètre linéaire des délais de communication. On remarque que le cas $\alpha = 0$ correspond au cas où toutes les communications prennent le même délai, et ce quelque soit la distance.

Lors des différents essais présentés dans la partie suivante, on travaille à délai moyen $\overline{\Delta T}$ équivalent, tout en modifiant le paramètre α pour obtenir des délais de communication plus ou moins variés.

À ces délais de communication s'ajoutent d'éventuels délais de retransmission dans le cas où le réseau de communication subit des pertes ou des dégradations de messages. En effet, l'utilisation d'un protocole comme TCP (*Transmission Control Protocol* [11]) s'assure que les messages soient bien arrivés à destination, et les renvoie si ce n'est pas le cas. Ainsi, dans un modèle simplifié de réseau de communication, la perte d'un message se traduit alors par l'ajout de délais supplémentaires.

On considère que la perte d'un message suit une variable de Bernoulli X de paramètre ψ avec $\mathbb{P}(X = 1) = \psi$ et $\mathbb{P}(X = 0) = 1 - \psi$. On considère cette variable aléatoire comme indépendante et identiquement distribuée sur tous les messages échangés lors de la résolution. Ainsi, lorsque $X = 1$, c'est-à-dire lorsque le message est perdu, il est renvoyé par l'expéditeur au bout d'un certain délai qui est supérieur au délai d'aller retour estimé par le protocole. Dans le cadre de notre modèle simplifié, on considère que la perte d'un message envoyé par l'agent i et destiné à l'agent j ajoute un délai supplémentaire de $2 \cdot \Delta T_{ij}$ au délai initial. Le message retransmis est lui même soumis au même processus de Bernoulli.

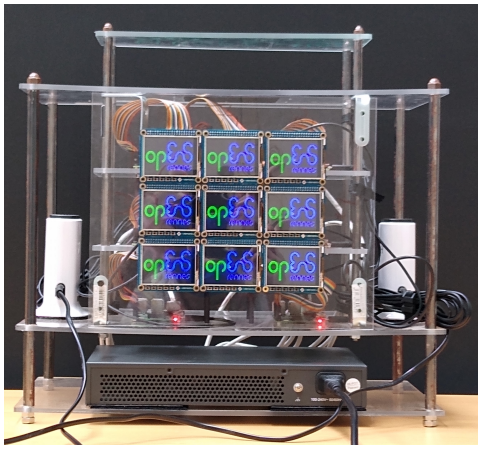

Fig. 3. Photo du cluster, constitué de 9 raspberry pis, de leurs écrans respectifs, ainsi que d'un commutateur.

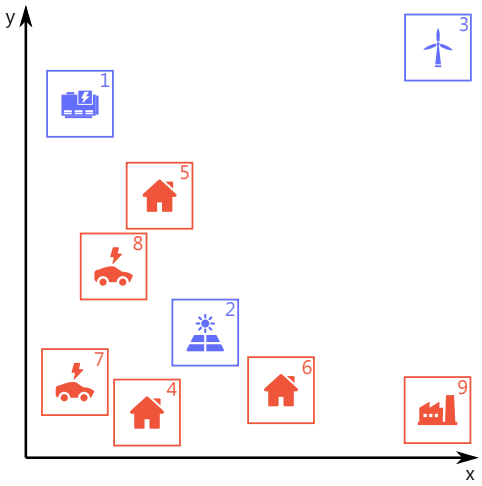

Fig. 4. Schéma des différents agents du marché et leurs numéros. Les producteurs sont représentés en bleu et les consommateurs en rouge. Leur position dans le plan permet de déterminer la distance D_{ij} mentionnée en partie 4.3 en calculant la distance euclidienne. On remarque dans notre cas d'étude que l'agent producteur 3 est éloigné des consommateurs, relativement aux autres producteurs.

4. IMPLÉMENTATION DE L'ALGORITHME

Dans la réalité, les agents participant au marché ne se situent pas sur le même réseau local, et doivent donc communiquer en s'envoyant des messages sur le réseau internet. Les protocoles utilisés doivent garantir une connexion bidirectionnelle et sans pertes entre les agents.

Dans le cas de notre application, il est intéressant de pouvoir suivre et enregistrer les échanges de message au fur et à mesure de la résolution par une seule entité. C'est pour cela que le protocole de communication choisi est le protocole MQTT. Il s'agit d'un protocole de messagerie *publish-subscribe*.

4.1. Agents implémentés sur Raspberry Pis

Nous disposons d'un ensemble de 9 Raspberry Pis connectés à un commutateur par liaison ethernet, comme illustré en Fig. 3. Ainsi, en implémentant un agent par Pi, nous avons $\Omega = \{1, 2, \ldots, 9\}$.

Les agents du marché considéré sont représentés en Fig. 4 [1]. Le marché est composé de trois agents producteurs dont un groupe électrogène (agent 1), une production photovoltaique (agent 2) et une production éolienne (agent 3). Tous ces agents sont flexibles au sens où la puissance qu'ils peuvent fournir au réseau de trouve entre 0 et $P_{i,max}$, avec $P_{i,max}$ pour $i \in \{2, 3\}$ dépendant de la puissance qu'il est possible de produire en fonction de l'ensoleillement et du vent au moment considéré. Le marché est aussi composé de six agents consommateurs dont trois maisons (agents 4,5 et 6), deux véhicules électriques (agents 7 et 8) et une industrie (agent 9). Les agents 4,5 et 6 sont non flexibles c'est-à-dire que leur consommation est fixée. Les agents 7, 8 et 9 sont des agents flexibles, à la différence près que l'agent 9 possède une consommation de base non flexible.

En ce qui concerne les liens commerciaux entre les agents, nous travaillerons sur un cas où tous les producteurs sont partenaires de tous les consommateurs et vice-versa. Ainsi, $\omega_i = \{4, 5, 6, 7, 8, 9\}$ pour $i \in \{1, 2, 3\}$ et $\omega_i = \{1, 2, 3\}$ pour $i \in \{4, 5, 6, 7, 8, 9\}$.

Le Tableau 2 donné dans l'annexe présente les données du problème utilisées par la suite. Les paramètres a_i et b_i représentent la fonction coût f_i de l'agent i telle que $f_i(x) = 2 \cdot a_i \cdot x^2 + b_i \cdot x$. Le facteur de pénalité de la résolution ADMM est réglé à $\rho = 2$, et le paramètre de convexité est réglé à $\gamma = 0.1$. Ce dernier est choisi de manière à ce que les résultats soient identiques en termes de trades pour une résolution synchrone et une résolution asynchrone, tout en modifiant le moins possible la solution en termes de puissance.

1. Crédits graphiques : Good Ware, Freepik, Vitaly Gorbachev et Thomas Knop sur flaticon.com

L'algorithme de chaque agent est implémenté en python, et prend en compte l'envoi et la réception de messages via un client MQTT ainsi que les mises à jour de l'algorithme 2.

Pour une utilisation pédagogique de la maquette, chaque Pi possède un écran de 2.4 pouces permettant d'afficher l'avancement de la résolution, ainsi que des boutons programmables qui permettent de modifier l'état de chaque agent pendant le déroulement de l'algorithme.

4.2. Communication MQTT

Le protocole de communication MQTT prend en compte deux types d'entités : un *broker* et des *clients*. Le broker est un serveur qui reçoit l'intégralité des messages envoyés par les clients et les renvoie aux clients destinés. Il s'agit d'un protocole léger, très souple et sécurisable. Un client se connecte au broker via le réseau de communication et lui annonce les *topics* auxquels il souhaite souscrire, c'est-à-dire les topics dont il souhaite recevoir les messages. N'importe quel client peut souscrire et publier sur n'importe quel topic.

Dans notre cas, le broker se trouve sur la machine principale, ce qui permet d'enregistrer les échanges de messages et de suivre la résolution de l'algorithme. Nous utilisons un broker open-source *Mosquitto*. À chaque agent est lié un client MQTT mis en oeuvre via l'API python *paho-mqtt*. Les échanges entre agents sont tous enregistrés dans une base de donnée *InfluxDB*. La suite de logiciels associés (Telegraf, Kapacitor et Chronograf) nous permet de visualiser en temps réel l'avancement de l'algorithme sur un dashboard.

4.3. Délais de communication et déclenchement des itérations locales

Étant donné la proximité des agents dans le réseau local, le délai de communication entre deux agents est de l'ordre de quelques millisecondes. Les délais de calcul sont eux de l'ordre de 20 millisecondes. Des délais additionnels sont ajoutés artificiellement pour simuler la distance sur un réseau entre les différents agents, selon le modèle de délais de communication présenté plus haut. Étant donné que les échanges de messages se font sur un vrai réseau, un terme stochastique apparaît dans les délais de communication réels mesurés.

Le délai moyen de communication $\overline{\Delta T}$ est fixé à une demi-seconde. Les délais de communication résultants sont

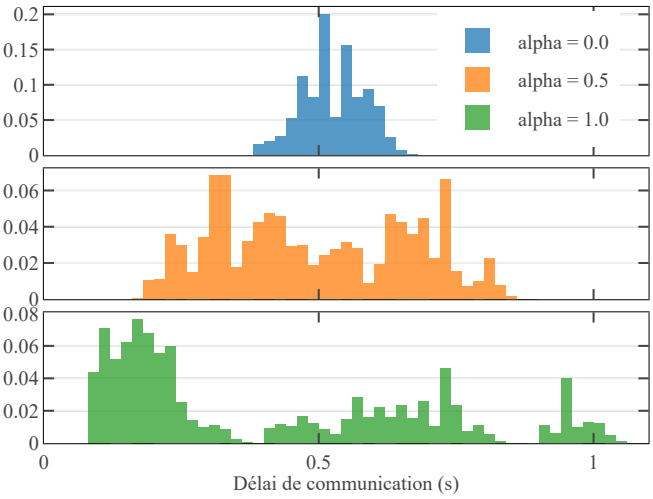

Fig. 5. Histogrammes des délais de communication pour différentes valeurs du paramètre linéaire du modèle de communication α, tels que le délai moyen de communication soit égal à $\overline{\Delta T} = 0.5s$, et sans retransmission $\psi = 0$.

représentés dans les histogrammes de la Fig. 5 dans le cas d'un réseau sans pertes de messages $\psi = 0$. Plus le paramètre α est grand, plus les délais de communication sont variés.

Le déclenchement d'une itération locale de l'agent i se fait après la réception de $N_{mess,i}$ nombre de messages. On introduit le paramètre d'asynchronisme $\delta \in [0, 1]$ tel que $\forall i \in \Omega$, $N_{mess,i} = \lceil \delta \cdot |\omega_i| \rceil$ où $\lceil \cdot \rceil$ est la fonction partie entière par excès. Le tableau Tableau 1 présente les différentes valeurs de $N_{mess,i}$ en fonction de δ pour les producteurs et les consommateurs. Lorsque $\delta = 1$, on se retrouve dans la configuration synchrone de l'algorithme.

Tableau 1. Nombre de messages provoquant le déclenchement d'une itération pour les producteurs et les consommateurs en fonction du paramètre d'asynchronisme δ

Paramètre d'asynchronisme δ	**$N_{mess,i}$ avec i producteur**	**$N_{mess,i}$ avec i consommateur**
1.0	6	3
0.8	5	3
0.6	4	2
0.4	3	2
0.1	1	1

5. RÉSULTATS

5.1. *Analyse de la convergence de l'algorithme*

L'avancement temporel de la résolution, représenté par le résidu primal global défini en (7a), est tracé en Fig. 6 pour différentes configurations du modèle de délais de communication présenté en partie 4.3, pour différentes valeurs de δ et dans le cas $\psi = 0$, c'est-à-dire sans perte de message. L'axe des abscisses correspond au temps relatif normalisé par le délai moyen $\overline{\Delta T}$ écoulé depuis le lancement de l'algorithme. Pour les mêmes essais effectués, on représente en Fig. 7 le résidu primal global en fonction non plus du temps, mais du nombre de messages totaux échangés entre les agents du marché.

On rappelle que le cas $\delta = 1$ (en bleu à marqueur rond sur la figure) est équivalent au cas synchrone de l'algorithme, c'est-à-dire que la durée d'une itération est au moins aussi longue que

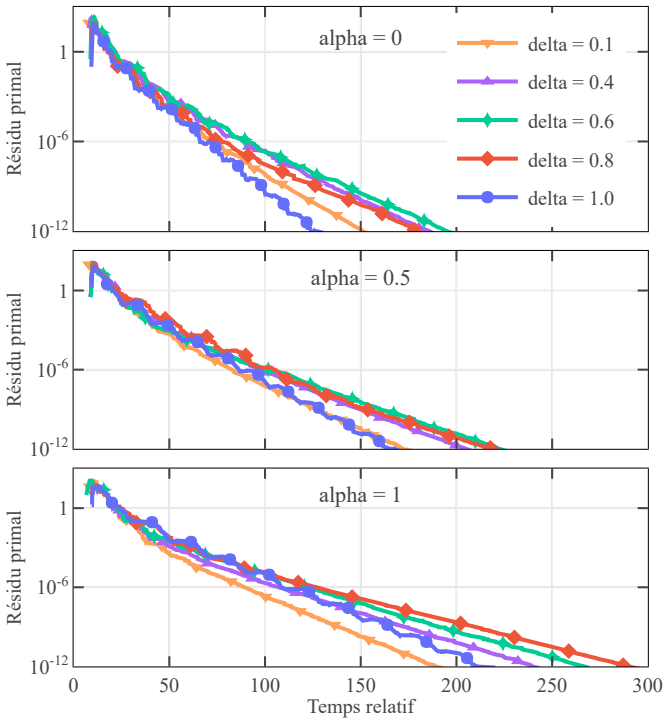

Fig. 6. Résidu primal global en fonction du temps relatif normalisé par $\overline{\Delta T}$, pour différentes configurations du modèle de délais de communication, et pour différentes valeurs du paramètre d'asynchronisme δ, avec $\psi = 0$.

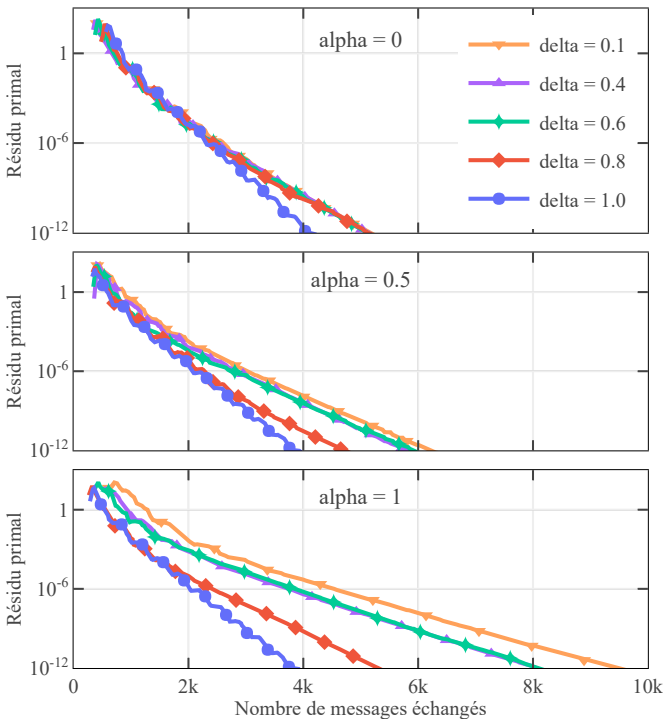

Fig. 7. Résidu primal global en fonction du nombre de messages échangés, pour différentes configurations du modèle de délais de communication, et pour différentes valeurs du paramètre d'asynchronisme δ, avec $\psi = 0$.

la plus longue durée de communication entre tous les agents du marché. Ainsi, plus le paramètre linéaire de délai de communication α est élevé, plus la durée de chaque itération est grande (comme on peut l'observer en Fig. 5) et plus l'algorithme est lent à converger.

Observons dans un premier temps le cas synchrone de la

Fig. 6. Il met en effet plus de temps à converger à mesure que le paramètre linéaire α augmente. Cependant, le nombre de messages échangés total reste le même à résidu primal global donné. On observe en Fig. 7 que le nombre de messages pour atteindre une convergence à $\varepsilon = 10^{-12}$ près est d'environ 4000 et ne dépend pas du paramètre α, ce qui est logique étant donné que les calculs effectués restent identiques.

Comparons maintenant les cas asynchrones au cas synchrone. Dans le cas où $\alpha = 0$ c'est-à-dire où les délais de communication sont identiques pour toutes les liaisons, le cas synchrone est non seulement plus rapide à converger, mais aussi utilise le moins de messages. En effet, puisque tous les délais sont égaux, tous les messages arrivent à destination à peu près au même instant et il y a peu de délai d'attente. Dans les cas asynchrones cependant, les calculs vont se déclencher avant que les agents aient reçu les derniers messages, qui seront donc traités lors de l'itération suivante.

Plus le paramètre α est élevé, plus les délais de communication sont variés, et selon la précision de convergence que l'on souhaite, certains cas asynchrones se montrent plus rapides que le cas synchrone. Si l'on veut une convergence très précise ($\varepsilon < 10^{-12}$), le cas synchrone prend en compte dans ses itérations le plus d'information possible, ce qui lui permet d'atteindre une grande précision plus rapidement. Cependant, dans le cas d'application de notre algorithme, il n'est pas nécessaire d'atteindre ce degré de précision et l'on se contente d'une convergence aux environs $\varepsilon = 10^{-3}$.

Parmi tous les cas asynchrones, le cas où le paramètre d'asynchronisme $\delta = 0.1$, celui où les agents effectuent une itération dès la réception d'un message, semble être le plus rapide en terme de convergence. Cette rapidité est atteinte au prix d'une augmentation des messages échangés. On observe en Fig. 7 pour $\alpha = 1$ qu'à précision équivalente, la version asynchrone $\delta = 0.1$ utilise plus de deux fois plus de messages que la version synchrone. Les autres cas asynchrones sont plus lents mais utilisent moins de messages que pour $\delta = 0.1$. Ainsi, le paramètre d'asynchronisme δ peut servir de degré de liberté pour trouver un compromis entre nombre de messages échangés et vitesse de convergence.

5.2. *Influence du taux de retransmission*

Jusqu'ici, les transmissions de messages entre agents étaient considérés parfaites, c'est-à-dire qu'il n'y avait pas de messages corrompus ou perdus. Considérons maintenant l'impact d'un réseau de communication avec pertes sur l'avancement de l'algorithme. Comme expliqué en partie 4.3, la perte d'un message se traduit par sa retransmission au bout d'un certain temps dans le cas où un protocole de communication comme TCP est utilisé. La probabilité de perte d'un message envoyé, appelé aussi le *taux de retransmission*, est notée ψ.

On peut imaginer que dans le cas synchrone, étant donné que chaque agent a besoin de tous les messages de tous ses partenaires commerciaux, le moindre délai ajouté se répercute directement sur la durée des itérations concernées. En revanche, dans le cas asynchrone, les agents sont capables de continuer leurs calculs même si un des messages attendus a du retard. Il sera alors pris en compte lorsqu'il arrivera à destination mais entre temps, l'agent destinataire aura continué ses calculs ainsi que ses échanges avec ses autres partenaires.

D'après les résultats de la Fig. 6, on observe que le cas où le paramètre d'asynchronisme $\delta = 0.1$ est le cas le plus rapide à converger parmi les cas asynchrones. Dans cette sous-partie, seul le cas $\delta = 0.1$ sera donc considéré.

Le but est de comparer les cas synchrone et asynchrone en termes de temps de convergence. La convergence de l'algorithme est considérée atteinte lorsque le résidu primal est en dessous d'une certaine valeur ε. Dans cette partie, cette valeur est fixée à $\varepsilon = 10^{-6}$. La Fig. 8 représente le temps de convergence

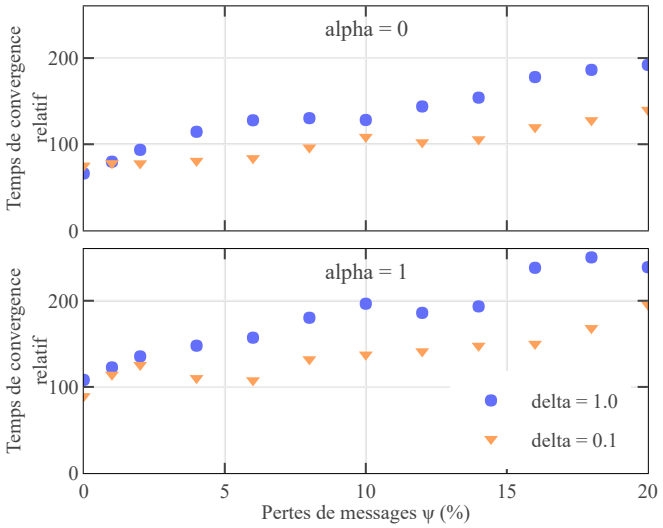

Fig. 8. Temps de convergence relatif, pour $\varepsilon = 10^{-6}$, en fonction du pourcentage de messages perdus et retransmis, pour les cas synchrone (en bleu) et asynchrone où $\delta = 0.1$ (en jaune), pour différentes valeurs du paramètre linéaire des délais de communication α.

relatif, normalisé par $\overline{\Delta T}$, en fonction du taux de retransmission ψ, pour les cas synchrone (en bleu) et asynchrone (en jaune).

On remarque pour $\psi = 0$ que ce gain est négatif dans les cas $\alpha = 0$, c'est-à-dire que l'algorithme synchrone est plus rapide que l'algorithme asynchrone, ce que l'on retrouve dans la Fig. 6. Dès que le réseau de communication admet $\psi \geq 1\%$ de pertes de messages, alors la version asynchrone de l'algorithme devient plus rapide que la version synchrone, et ce quelque soit la variabilité des délais de communication, représentée par le paramètre α. Ici, les données sont extraites à partir d'une seule expérience par cas étudié. Pour pouvoir conclure sur les gains de temps de convergence, il faudrait répéter ces expériences de nombreuses fois pour pouvoir déduire des statistiques à partir de la méthode de Monte-Carlo. Toutefois, étant donné le nombre élevé de messages échangés, donc le nombre de tirages aléatoires et indépendants du nombre de messages retransmis, on peut considérer que le résultat obtenu est proche de la moyenne.

5.3. *Déconnexion d'un agent du réseau de communication*

Cette partie s'intéresse au cas où un agent est momentanément déconnecté du réseau de communication. Le cas d'étude considéré consiste en la déconnexion de l'agent 7 au bout de trente secondes après le commencement de la résolution, et sa reconnexion au réseau quinze secondes plus tard. On se place dans le cas où $\alpha = 0$, c'est-à-dire où les délais de communication sont tous égaux à $0.5s$, et $\psi = 0$ c'est-à-dire sans perte de message.

Les cas synchrone et asynchrone pour différentes valeurs du paramètre d'asynchronisme δ est considéré. La Fig. 9 représente le résidu primal en fonction du temps.

Pour le cas synchrone (en bleu à marqueur rond sur la figure), le résidu primal global se fige pendant toute la durée de déconnexion, tandis que pour les cas asynchrones, les agents ne communiquent plus avec l'agent déconnecté mais peuvent toujours communiquer entre eux et continuer à faire décroître le résidu primal global. On remarque cependant un ralentissement de cette diminution, qui peut être expliqué par le fait que la partie du résidu primal de chaque agent partenaire de l'agent 7 : $t_{i7} + t_{7i}$ ne décroît pas.

À la reprise normale des communications, le cas synchrone reprend alors là où il s'était arrêté. Dans cette configuration du réseau de communication, la version synchrone est bien plus

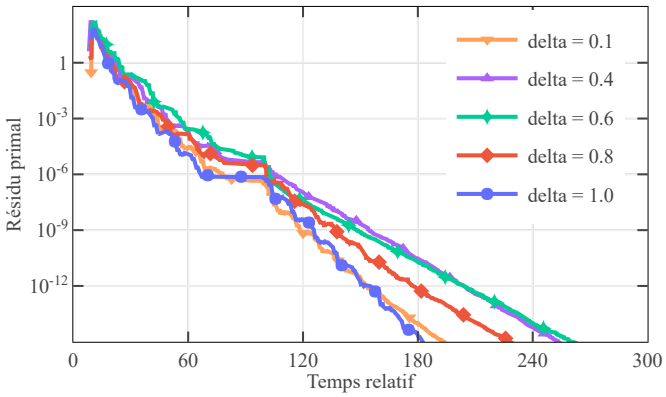

Fig. 9. Résidu primal global en fonction du temps relatif normalisé par $\overline{\Delta T}$, et pour différentes valeurs du paramètre d'asynchronisme δ, avec $\psi = 0$ et $\alpha = 0$.

rapide que les versions asynchrones ce qui explique que le résidu primal reste en général plus petit qu'en asynchrone. Cependant, on remarque que le cas asynchrone $\delta = 0.1$ est équivalent au cas synchrone là où il était plus lent en Fig. 6 là où il n'y avait pas de déconnexion d'un agent.

6. CONCLUSION

Nous avons étudié dans cet article la vitesse de convergence d'un algorithme décentralisé pair à pair de résolution de marché, étudié dans le cadre d'un réseau de communication avec délais et pertes. L'implémentation réelle de l'algorithme limitant le nombre d'agents du marché, nous avons travaillé avec un marché réduit à trois producteurs et six consommateurs. Cette implémentation, en plus de permettre de tester l'algorithme dans différentes configurations, a aussi une portée pédagogique.

Les performances des versions synchrone et asynchrone de l'algorithme ont été étudiées en fonction de l'état du réseau de communication. Il s'avère que pour un marché aussi petit et tant que le réseau de communication est fiable, la version synchrone de l'algorithme est en général plus rapide à converger que la version asynchrone. En revanche, dès qu'il commence à y avoir des pertes de messages, nous avons montré que la version asynchrone était plus rapide. De plus, dans le cas d'une déconnexion momentanée d'un des agents du marché, la version asynchrone permet à l'algorithme de continuer ses calculs contrairement à la version synchrone. On peut conclure que la version asynchrone rend l'algorithme plus résilient par rapport à l'état du réseau de communication. Cependant, le nombre de messages échangés augmente significativement. Le paramètre d'asynchronisme δ permet de trouver un compromis entre nombre de messages échangés et vitesse de convergence.

Dans les perspectives de cette étude, il serait intéressant dans un premier temps d'ajouter des contraintes physiques liées aux limitations du réseau électrique pour obtenir des résultats directement applicables sur le réseau. On pourrait aussi étudier le cas d'un marché avec signicativement plus d'agents pour vérifier la mise à l'échelle de l'algorithme. Aussi, nous avons étudié dans cette article des versions asynchrones où le paramètre d'asynchronisme δ était identique à tous les agents. Il pourrait être intéressant d'étudier des cas où chaque agent choisit un paramètre δ différent, à travers un apprentissage en ligne, dans le but d'être le plus rapide tout en échangeant le moins de messages possibles.

7. RÉFÉRENCES

[1] J. Zhang, S. Nabavi, A. Chakrabortty et Y. Xin, « ADMM Optimization Strategies for Wide-Area Oscillation Monitoring in Power Systems Under Asynchronous Communication Delays », in IEEE Transactions on Smart Grid, vol. 7, no. 4, p. 2123 2133, Juillet 2016.

[2] J. Zhang, S. Nabavi, A. Chakrabortty et Y. Xin, « Convergence analysis of ADMM-based power system mode estimation under asynchronous wide-area communication delays », 2015 IEEE Power & Energy Society General Meeting, Denver, CO, 2015, p. 1 5.

[3] R. Zhu, D. Niu et Z. Li, « A Block-wise, Asynchronous and Distributed ADMM Algorithm for General Form Consensus Optimization », ArXiv, 2018

[4] M. H. Ullah et J. Park, « Distributed Energy Optimization in MAS-based Microgrids using Asynchronous ADMM », 2019 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT), Washington, DC, USA, 2019, p. 1 5.

[5] J. Guo, G. Hug et O. Tonguz, « Impact of communication delay on asynchronous distributed optimal power flow using ADMM », 2017 IEEE International Conference on Smart Grid Communications (SmartGridComm), Dresden, 2017, p. 177 182.

[6] J. Zhang, « Asynchronous Decentralized Consensus ADMM for Distributed Machine Learning », 2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS), Shenzhen, China, 2019, p. 22 28.

[7] A. Huang, S-K Joo, K.B. Song, J.H. Kim, et K. Lee, « Asynchronous decentralized method for interconnected electricity markets », International Journal of Electrical Power and Energy Systems, vol. 30, no. 4, p . 283 290, 2008.

[8] R. Zhang et J. T. Kwok, « Asynchronous distributed ADMM for consensus optimization », In Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14). JMLR.org, 2014.

[9] T. Baroche, P. Pinson, R. Le Goff Latimier et H. Ben Ahmed, « Exogenous Cost Allocation in Peer-to-Peer Electricity Markets », in IEEE Transactions on Power Systems, vol. 34, no. 4, p. 2553 2564, Juillet 2019.

[10] S. Boyd, N. Parikh, E. Chu, B. Peleato, et J. Eckstein, « Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers », MAL, vol. 3, n° 1, p. 1 122, Juillet 2011.

[11] «Transmission Control Protocol», Request for Comments n°793, Septembre 1981.

8. ANNEXE

Le Tableau 2 représente les paramètres des agents liés à leur fonction coût $f_i(x) = 2 \cdot a_i \cdot x^2 + b_i \cdot x$ ainsi qu'à leur contrainte de production/consommation $\underline{p_i} \leq p_i \leq \overline{p_i}$.

Tableau 2. Données des agents

i	$\mathbf{a_i}$	$\mathbf{b_i}$	$\underline{\mathbf{p_i}}$	$\overline{\mathbf{p_i}}$
1	0.03	1.2	0	200
2	0.04	-24	0	20
3	0.04	-24	0	27
4	0.0	0.0	-8	-8
5	0.0	0.0	-6	-6
6	0.0	0.0	-15	-15
7	0.03	1.2	-15	0
8	0.03	1.2	-15	0
9	0.02	-8.0	-100	-15