Multidistribution Multi-sources Adaptative

Bertrand Ducourthial et Sebastien Tixeuil

Laboratoire HEUDIASYC, UMR CNRS 6599, Université de Technologie de Compiègne, 60205 Compiègne CEDEX
email: Bertrand.Ducourthial@utc.fr
Laboratoire de Recherche en Informatique, UMR CNRS 8623, Université Paris-Sud, 91405 Orsay CEDEX
email: tixeuil@lri.fr

Dans cet article, nous nous intéressons à la détermination d'un schéma de multidistribution [3] permettant les meilleures retransmissions, dans le cas où le critère d'évaluation peut être exprimé localement au niveau de chaque lien, sous la forme d'une métrique habituelle (poids, taux, *etc.*). Nous proposons un algorithme de construction d'une forêt de multidistribution qui est auto-stabilisant, c'est-à-dire qui résiste aux défaillances temporaires des sites ou des liens de communication, et qui tolère des aléas de communication (pertes, duplications, déséquencements). La propriété d'auto-stabilisation lui permet, sous certaines conditions, de s'adapter à une nouvelle configuration (topologie, variation des poids ou des taux, modification des groupes) sans avoir à écrire de code spécifique.

Keywords: multidistribution multi-sources, algorithme adaptatif, auto-stabilisation

1 Introduction

1.1 Multidistribution multi-source

Nous nous plaçons dans le cadre d'un réseau de communication dans lequel une ou plusieurs sources détiennent les mêmes données, dont une partie (éventuellement nulle) est désirée par les autres sites, organisés en groupes (*cf.* [3]). À chaque unité de données détenue par les sources correspond un groupe de destinataires, éventuellement vide. Un site appartient à autant de groupes qu'il désire de données ; à tout moment un site peut rejoindre ou quitter un groupe. Cette hypothèse de travail correspond par exemple à la mise à jour de sites miroirs de serveurs *ftp*, *nntp* ou *http*. Elle correspond aussi à la diffusion de films à la demande : des sources diffusent plusieurs films ; les autres sites ne désirent qu'un sous-ensemble de ces films, voire aucun. Nous souhaitons distribuer dans le réseau les données en provenance des sources, de sorte qu'aucune donnée ne soit diffusée inutilement, et que pour chaque site, les données reçues aient suivi un meilleur chemin depuis une meilleure source, selon un critère relatif au chemin (taux de fiabilité, taux de perte de paquets, nombre de nœuds intermédiaires, longueur, débit...).

1.2 Multidistribution adaptative

La fiabilité des composants de base (processeurs et liens) n'étant pas parfaite, divers défaillances peuvent survenir au cours de la distribution. Des corruptions de mémoire sur les processeurs peuvent apparaître suite à une panne suivie d'une réparation, ou bien lors d'une défaillance temporaire de l'alimentation électrique. Pour un lien de communication, des déconnexions momentanées conduisent à des problèmes similaires. Dans un cadre réparti, où les processeurs n'ont qu'une vision locale du système, il est souvent impossible de déterminer qu'une telle défaillance a eu lieu, et ces corruptions de mémoire sont susceptibles de mener le système dans un état arbitraire (*cf.* [18]). Nous souhaitons obtenir un schéma de multidistribution adaptatif, c'est-à-dire d'une part qui tolère les défaillances temporaires des éléments du réseau, et d'autre part qui supporte les évolutions des critères qui déterminent les meilleurs chemins (évolution de la topologie, du taux de perte de paquets, *etc.*).

1.3 État de l'art

Pour obtenir une multidistribution adaptative, nous utilisons le concept de l'auto-stabilisation, introduit par Dijkstra en 1974 [2, 6, 17]. Un système réparti auto-stabilisant a la faculté de retrouver sans intervention extérieure un comportement correct après un temps fini (et dans le cas présent borné par le diamètre du réseau), et ce quelque soit l'état du système initialement. L'auto-stabilisation est particulièrement utile lors de calculs globaux distribués (*cf.* [4]), tels la construction de forêts couvrantes, puisqu'un algorithme vérifiant cette propriété est naturellement adaptatif aux entrées réparties sur les processeurs du système. Dans le cadre particulier de la construction d'arbres et de forêts vérifiant des critères particuliers, on peut citer les travaux de Dolev et Herman [5] (pour les arbres de plus courts chemins), de Schneider et Gouda [14] (pour les arbres de flot maximal), et de Gupta, Bouabdallah et Srimani [15, 16] (pour les arbres de multidistribution).

Dans [15, 16], une solution au problème de la multidistribution est proposée ; elle est adaptative aux évolutions de la topologie. Cette solution comporte deux phases : *(i)* construction auto-stabilisante d'un arbre de hauteur minimale à partir d'une source de données, et *(ii)* élagage de l'arbre ainsi construit pour tenir compte des sites qui ne désirent pas participer à la multidistribution depuis la source. La distribution est effectuée à partir d'un seul site, la racine de l'arbre. L'arbre de diffusion est déterminé selon un critère additif, à savoir les poids des liens, qui sont supposés fixés et connus à l'avance. Un seul groupe de distribution est considéré. Les liens de communication sont supposés fiables et FIFO (pas de déséquencement).

1.4 Notre Contribution

Nous proposons un schéma de multidistribution optimale (du point de vue du critère employé), multi-sources, multi-groupes, et adaptatif. Cette solution présente les avantages suivants :

(i) Nous maintenons une forêt couvrante optimale selon un critère donné en utilisant l'approche de [9]. La racine de chaque arbre de cette forêt est une source potentielle de données pour les sites intéressés. Nous autorisons ainsi l'utilisation de sites miroirs, qui permettent de relayer l'information de manière plus fiable ou plus rapide vers certaines parties du réseau.

(ii) Le schéma de distribution est adaptatif et supporte, sous certaines conditions, les variations des critères définissant les chemins. Il n'est plus nécessaire de mettre à jour le code du programme à chaque évolution du traffic, ou modification de l'infrastructure, dûe au changement d'un composant (site ou lien) ou à une panne temporaire.

(iii) Les métriques supportées peuvent être de plusieurs types. Par exemple, les liens de communication peuvent être caractérisés par un coût (additif le long d'un chemin), ou bien par un taux de fiabilité (multiplicatif le long d'un chemin).

(iv) Plusieurs groupes de multidistribution sont gérés. Un site peut appartenir à aucun, un ou plusieurs groupes. Les données ne sont délivrées qu'aux sites destinataires, *ie.* membres du groupe correspondant (dans certains cas cependant, il est possible que des nœuds intermédiaires retransmettent des données dont ils ne sont pas demandeurs). Ces groupes sont définis dynamiquement. Une optimisation est effectuée en cas de multidistribution à plusieurs groupes de la même donnée, même dans le cas où un site particulier est membre de plusieurs groupes relayant cette donnée.

(v) Des défaillances intermittentes des liens de communication sont supportées : perte équitable (si une infinité de messages sont émis, alors au moins un message est reçu), duplication finie (chaque message, erroné ou non, peut être dupliqué un nombre fini de fois) et déséquencement arbitraire (les canaux ne sont pas nécessairement FIFO).

(vi) Les messages de contrôle de notre protocole n'ont pas besoin d'être acquittés. Notre algorithme peut donc être implanté avec un protocole de communication efficace, tel que UDP.

2 Système réparti

2.1 Algorithme générique paramétré

Nous modélisons le réseau par un graphe orienté $G(V, E)$, où les nœuds représentent les processeurs et les arcs les liens de communication. Nous notons δ_v^- le nombre d'arcs entrants du nœud v. Les communications entre les processeurs s'effectuent par échange de messages. Aucune hypothè particulière n'est supposée quant à la fiabilité des liens de communication : ils peuvent perdre équitablement des message, en dupliquer un nombre fini, et changer l'ordre relatif des messages en transit entre leur émission et leur réception. Dans [13], Gouda et Multari ont montré que tout algorithme auto-stabilisant dont les communications s'effectuent par passage de message doit être en mesure d'émettre régulièrement des messages. Notre algorithme satisfait ce critère.

Chaque nœud v possède :

(i) une variable privée X_v ;

(ii) un tableau $\mathrm{TabX}_v[1..\delta_v^-]$ d'autant de cases que de liens entrant sur le nœud ;

(iii) une variable XRes_v accessible en lecture par ses descendants dans le réseau.

Nous supposons en outre que la variable X_v de chaque nœud v ne peut rester indéfiniment corrompue : elle est stockée en ROM, ou régulièrement réinitialisée par la couche applicative du site.

Nous utilisons par la suite un algorithme générique simple, paramétré localement par un opérateur (*cf.* [7, 8], ou [9, 10] pour une présentation dans le modèle de communication par registres). Son principe est le suivant :

1. le tableau TabX_v d'un nœud v est régulièrement mis à jour avec les variables XRes_u des antécédents directs u de v dans le réseau ;

2. à chaque fois que nécessaire, le nœud v met à jour sa variable XRes_v avec le résultat du calcul par un opérateur f_v appliqué à la variable X_v et à celles du tableau TabX_v :

$$\mathrm{XRes}_v \leftarrow f(\mathrm{X}_v, \mathrm{TabX}_v[1], \ldots, \mathrm{TabX}_v[\delta_v^-])$$

La phase de mise à jour *(1)* nécessite que des communications soient entreprises. Afin de pallier aux défaillances transitoires qui peuvent survenir, il est nécessaire d'envoyer régulièrement le contenu de la variable XRes_v de chaque nœud v à ses descendants dans le réseau. Ce mécanisme permet de propager les mises à jour des variables XRes à travers le réseau. De même, pour implanter la phase de calcul *(2)*, il est suffisant de recalculer régulièrement le résultat de l'opérateur f appliqué à la variable X et au tableau TabX.

2.2 Conditions de stabilisation

Les résultats de G. Tel sur les infimums [20] assurent que l'algorithme précédemment décrit stabilise lorsque les opérateurs — appelés par la suite s-opérateurs — sont binaires, associatifs, commutatifs et idempotents. Dans [7], des opérateurs plus généraux — appelés r-opérateurs — ont été introduits ; ils conduisent également à la stabilité du réseau.

Nous avons montré [1, 9] que lorsqu'il est paramétré par un s-opérateurs, l'algorithme générique n'est auto-stabilisant que sur des réseaux sans circuit. Dans [9, 10], nous avons montré que si l'algorithme était paramétré par un r-opérateur strictement idempotent, alors il était auto-stabilisant quelle que soit la topologie du réseau. Ce résultat a été montré dans le cadre de communication par registres. Nous l'adaptons ici au modèle de communication par passage de messages en utilisant la constante remise à jour des tableaux TabX à partir des variables XRes des antécédants directs. Dans [9], les hypothèses de synchronisation des nœuds sont supposées les plus faibles pour le moèle considéré (démon lecture-écriture). Dans [10], elles sont plus contraignantes (démon complètement réparti) mais le champs d'opérateurs conduisant à la terminaison est plus vaste.

Nous résumons ci-dessous les conditions suffisantes portant sur le paramètre f de l'algorithme, de telle sorte que ce dernier soit auto-stabilisant sur tout type de topologie, malgré une atomicité faible (lecture-écriture) :

(a) pour un entier $n > 1$ donné, $f : \mathbb{S}^n \mapsto \mathbb{S}$ est défini par : $f(x_0, \dots, x_n) = x_0 \oplus r_1(x_1) \oplus \cdots \oplus r_n(x_n)$;

(b) l'ensemble de définition $\mathbb{S}$ est fini, ou toute suite strictement croissante est non bornée ;

(c) $\oplus$ est un s-opérateur qui induit une relation d'ordre total $\preceq_\oplus$ sur $\mathbb{S}$ (*i.e.* $x \preceq_\oplus y \equiv x \oplus y = x$) ; on note $e_\oplus$ le plus grand élément de $\mathbb{S}$;

(d) les fonctions r_i sont des homomorphismes de $(\mathbb{S}, \oplus)$ ($r_i(x \oplus y) = r_i(x) \oplus r_i(y)$) ;

(e) pour toute fonction r_i et pour tout $x \in \mathbb{S} \setminus \{e_\oplus\}$, on a $x \prec_\oplus r_i(x)$ (*i.e.* $x \preceq_\oplus r_i(x)$ et $x \neq r_i(x)$).

Les conditions *(a)*, *(c)*, *(d)* et *(e)* définissent un r-opérateur strictement idempotent. La condition *(b)* et l'hypothèse de l'ordre *total*, ne sont imposées que par le choix d'une atomicité faible. On vérifie ainsi que l'opérateur minc défini par $\mathrm{minc}(x, y) = \min(x, y + 1)$ fournit un algorithme auto-stabilisant [8].

2.3 Résultat

Lorsque les conditions *(a)* à *(e)* ci-dessus sont remplies, ou bien lorsqu'il n'y a pas de circuit dans le réseau, l'algorithme stabilise et le résultat XRes_v de chaque nœud v est égal au minimum (au sens de $\preceq_\oplus$) des valeurs X_u de chaque antécédent u, transformées par les fonctions r_i le long des chemins reliant l'antécédant u au nœud v. Par exemple, lorsque l'algorithme est paramétré par l'opérateur minc, que $\mathrm{X}_u = 0$ pour certains nœuds u sources, que $\mathrm{X}_v = \infty$ (*ie.* plus grande valeur autorisée) pour les autres nœuds, alors l'algorithme stabilise et chaque nœud obtient la distance à la source la plus proche [9].

2.4 Hypothèses de communication

Puisque la variable XRes_u d'un nœud u est régulièrement recopiée sur chaque descendant v de u dans le réseau, les pertes équitables concernant les messages émis sur le lien (u, v) n'auront pas de conséquence sur le déroulement de l'algorithme. Un lien de communication perd équitablement les messages si, une infinité de messages étant émis sur ce lien, une infinité de messages est reçue de ce même lien.

Puisqu'aucune hypothèse n'est faite sur les vitesses relatives des différents processeurs du système, il est tout à fait possible qu'un processeur ait eu le temps de transmettre k fois le même message avant que l'un de ses descendants directs ait eu le temps de le lire une seule fois. De ce fait, les duplications finies sont assimilées à des différences relatives entres les vitesses d'exécution des différents processeurs. En conséquence l'algorithme totalement asynchrone que nous proposons peut les supporter.

Enfin, notre algorithme permet de considérer chaque message comme une entité indépendante, dont le contenu (la valeur) est modifié suivant les résultats des opérations qui sont effectuées par les processeurs. Plus précisément, nous prouvons que chaque message pris isolément ne peut rester incorrect indéfiniment. C'est cette indépendance qui permet à notre algorithme aux déséquencements éventuels qui pourraient survenir lors de l'utilisation d'un protocole de communication efficace comme UDP.

Concernant chacun de ces aspects, il est remarquable que notre algorithme n'effectue aucun contrôle ou traitement spécifique (comme l'utilisation de numéros de séquence pour gérer les déséquencements) pour gérer ces défaillances intermittentes des liens de communication. En particulier, le fait que la tolérance aux pertes de messages s'effectue *sans* l'utilisation habituelle d'acquittements des messages reçus, il est possible d'utiliser notre algorithme dans des réseaux où les communications ne sont pas nécessairement bidirectionelles. Ceci est par exemple le cas dans le cas d'emissions/réception par ondes radio, ou le rayon de réception est rarement le même que le rayon d'émission.

2.5 Adaptativité

Sous réserve des conditions de stabilisation, l'algorithme générique paramétré se stabilise sur un résultat lorsque ses entrées n'évoluent plus. Ainsi, en cas de modification inopinée d'une variable sur un nœud, l'algorithme redevient actif et corrige l'erreur transitoire en temps fini (proportionnel au diamètre du réseau de communication).

Parmi les entrées de l'algorithme, il y a également l'opérateur f. Ce paramètre de l'algorithme peut lui-même dépendre de variables susceptibles d'évoluer. Par exemple, si le critère de construction de la forêt est la distance, on utilisera un opérateur similaire à l'opérateur minc tel que $\min(x, x_1 + d_1, \ldots x_n + d_n)$, qui est défini à l'aide des variables $d_1 \ldots d_n$, les longueurs des liens entrants sur le nœud. Si le réseau est constitué de nœuds mobiles, les variables $d_1 \ldots d_n$ sont susceptibles d'évoluer dans le temps, entrainant la modification de l'opérateur (le paramètre f). L'algorithme générique paramétré s'adapte aux évolutions du réseau en ce sens qu'il stabilise lorsque l'opérateur-paramètre évolue en respectant l'une des conditions suivantes :

(i) il n'y a pas de corrélation entre le choix d'une forêt de distribution et l'évolution du critère des chemins, ou

(ii) l'évolution des critères est plus lente que le temps de convergence de l'algorithme ($O(D)$).

3 Multidistribution

Notre approche utilise un algorithme à deux phases. Dans un premier temps, nous construisons de manière auto-stabilisante une forêt de distribution optimale suivant la méthode décrite dans [9]. Dans un second temps, nous construisons une sous-forêt de multidistribution à l'intérieur de cette première forêt optimale, de manière à distribuer les données en fonction de l'appartenance des nœuds aux différents groupes. Ces deux algorithmes sont par exemple composés suivant l'approche décrite dans [12], où le résultat d'un premier algorithme est utilisé en entrée d'un second algorithme, ces deux algorithmes étant exécutés en parallèle. Dans les deux cas, nous utilisons l'algorithme générique paramétré défini précédemment. Notons que lors de la première phase, le réseau peut présenter des circuits, ce qui n'est pas le cas lors de la deuxième phase.

3.1 Utilisation de l'algorithme générique paramétré

Chaque nœud v possède les variables F_v, G_v, FRes_v et GRes_v, ainsi que les tableaux TabF_v et TabG_v de δ_v^- cases. La variable F_v indique si le nœud v est ou non une source. Cette variable est stockée en ROM, ou bien régulièrement réinitialisée par la couche applicative, de sorte qu'elle ne reste pas infiniment longtemps corrompue. La variable G_v indique l'appartenance du nœud v aux groupes de diffusion. Elle peut être modifiée à tout moment par la couche applicative du site. On utilise deux algorithmes génériques, un par phase. Le premier utilise les variables F_v, TabF_v, FRes_v et un opérateur f_v (voir ci-après) et permet de construire la <u>F</u>orêt. Le second utilise les variables G_v, TabG_v, GRes_v et un opérateur g_v, et permet de construire les <u>G</u>roupes de multidistribution.

L'utilisation de l'algorithme générique paramétré assure, pour chaque nœud dans le réseau, d'une part que les tableaux TabF_v et TabG_v sont régulièrement mis à jour avec les valeurs des variables FRes_u et GRes_u des antécédents directs u de v, et d'autre part que les variables FRes_v et GRes_v sont mises à jour avec le résultat des calculs $f(\mathrm{T}_v,\mathrm{TabT}_v[0],...,\mathrm{TabT}_v[\delta_v^- - 1])$ et $g(\mathrm{G}_v,\mathrm{TabG}_v[0],...,\mathrm{TabG}_v[\delta_v^- - 1])$, respectivement.

3.2 Construction de la forêt de diffusion optimale

Lors de cette phase, le réseau sous-jacent peut comporter des circuits. Il est donc nécessaire que l'opérateur f soit un r-opérateur strictement idempotent et que les conditions *(a)* à *(e)* soient vérifiées.

Un large spectre d'opérateurs — et donc de forêts — sont autorisés. Parmi les opérateurs conduisant aux forêts les plus communes, citons les opérateurs du type de $(x, y) \mapsto \min(x, y + 1)$ pour les critères additifs (forêt des plus courts chemins, en distance ou en nombre de nœuds intermédiaires), et les opérateurs du type $(x, y) \mapsto \max(x, y \times \tau)$ pour les critères multiplicatifs (forêt des meilleurs taux de fiabilité, de pertes de paquets, *etc.*).

Suivant [11], une forêt des meilleurs taux de fiabilité sera obtenu de la manière suivante. On échantillonne les taux des liens sur $2^k - 1$ valeurs, depuis 0 (ligne hors-service) jusqu'à $2^k - 1$ (ligne parfaite). Le taux d'un chemin P sera obtenu en effectuant le produit des taux des arcs le composant. Soit τ_v^i le taux du $i^{\text{ème}}$ arc incident du nœud v. Nous définissons l'opérateur maxmul_τ par :

$$\mathrm{maxmul}_\tau(x_0, \ldots , x_{n-1}) = \max\left(x_0, \lfloor x_1 \times \tau_v^1/(2^k - 1)\rfloor, \ldots , \lfloor x_{n-1} \times \tau_v^{n-1}/(2^k - 1)\rfloor\right)$$

L'opérateur maxmul_τ étant un r-opérateur, nous avons la garantie que [8, 9] :

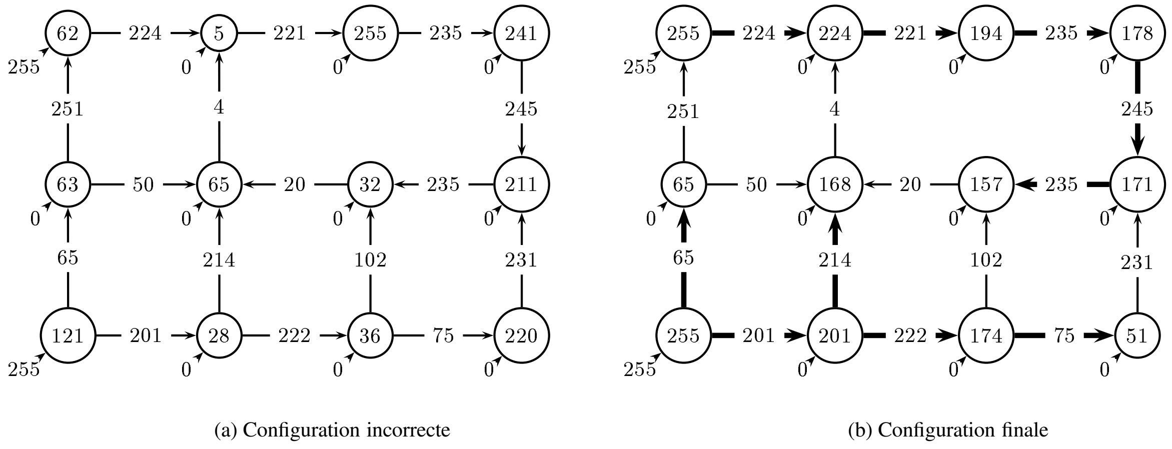


Fig. 1: Construction de la forêt de diffusion optimale

(i) le calcul termine en $O(D)$ et chaque site obtient le meilleur taux possible depuis une source du réseau, et

(ii) des défaillances intermittentes de liens (perte, duplication, déséquencement) peuvent être supportées pour ces messages de contrôle.

La figure 1 présente deux configurations possibles d'un système où les taux de fiabilité des liens de communications sont échantillonés de 0 (lien hors service) à 255 (lien totalement fiable). Un cercle de la figure représente un processeur: une valeur située en dehors du cercle représentant le nœud v dénote la variable X_v, une valeur située à l'intérieur du cercle dénote la variable XRes_v. Dans la configuration finale, les arcs correspondant à la forêt construite par notre algorithme sont représentés en gras.

À l'issue de cette phase, chaque site v possède une variable P_v qui pointe vers son nœud parent dans la forêt optimale de diffusion : cette variable est donc également adaptative aux entrées.

3.3 Construction des groupes de multidistribution

Lors de cette phase, le réseau sous-jacent est la forêt de distribution construite à l'étape précédente, mais orientée des feuilles vers les sources. Ce réseau ne comporte donc pas de circuit et tout r-opérateur (y compris les s-opérateurs, où les r-opérateurs non-idempotents) peuvent être employés.

Les groupes seront obtenus de la façon suivante. Les opérateurs des nœuds utilisent des registres de l bits. Ceci permet de gérer des ensembles d'au plus l groupes de multidistribution, et d'effectuer des opérations d'union ensembliste au moyen d'un *ou* bit à bit (noté $|$) et d'intersection au moyen d'un *et* bit à bit (noté $\&$).

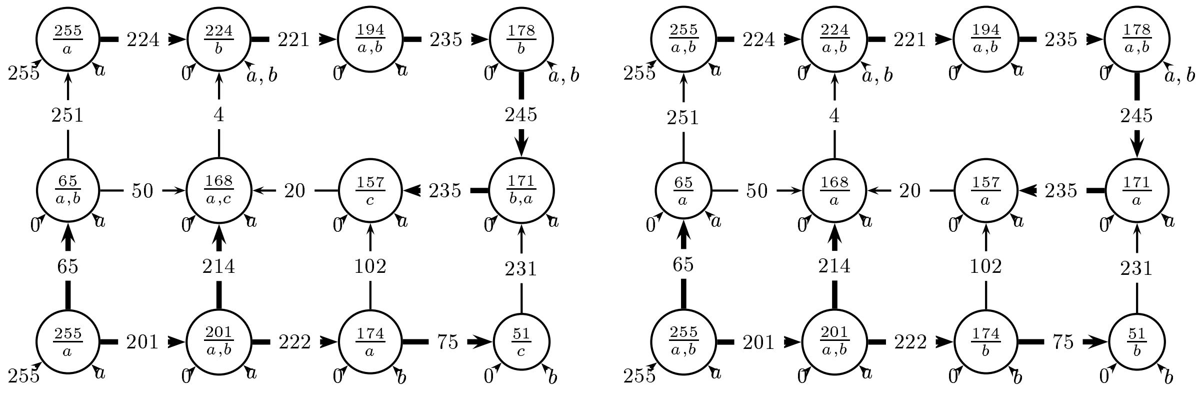
Pour déterminer l'ensemble des groupes de multidistribution qui sont présents dans le sous-arbre dont il est la racine, un site effectue simplement l'union des groupes présents dans chacun de ses sous-arbres (donc l'union des ensembles calculés par chacun de ses fils) et de son propre ensemble de groupes. Soient $TabG_{v[1]} \ldots TabG_{v[\delta_v^-]}$ les ensembles de groupes de multidistribution des fils de v dans la forêt. Nous définissons l'opérateur union_v par :

$$\mathrm{union}_v(G_v, TabG_{v[1]}, \ldots, TabG_{v[\delta_v^-]}) = G_v|TabG_{v[1]}| \ldots |TabG_{v[\delta_v^-]}$$

La figure 2 représente deux configurations de l'algorithme de construction des groupes de multidistribution composé avec l'algorithme de construction de forêts de fiabilité maximale. Dans la configuration incorrecte, nous avons supposé que le résultat du premier algorithme était acquis. C'est pourquoi aucune corruption n'apparaît concernant les variables relatives à l'algorithme de construction de forêts.

L'opérateur union_v étant un s-opérateur, et le réseau sous-jacent étant sans circuit, nous avons la garantie que ([1, 8, 9]) :

(i) le calcul termine et chaque site obtient finalement le bon ensemble de groupes présents dans le sous-arbre dont il est la racine ;



(a) Configuration incorrecte

(b) Configuration finale

Fig. 2: Construction des groupes de multidistribution

(ii) des défaillances intermittentes de liens peuvent être supportées pour ces messages de contrôle ;

(iii) la terminaison de cet opérateur est assurée en un temps $O(D)$, où D est le diamètre du réseau.

3.4 Utilisation des groupes

On suppose que chaque message d'information est composé de deux parties : $\langle G, M\rangle$, où G désigne l'ensemble des groupes qui sont destinataires de ce message (le plus souvent, G est réduit à un singleton), et où M désigne la donnée à transmettre. Pour utiliser effectivement la construction de groupes de multidistribution précédemment effectuée, il suffit de procéder comme suit sur chaque site :

(i) Chaque nœud v de la forêt couvrante de diffusion vérifie, sur réception d'un message $\langle G, M\rangle$, que $G_v \cap G$ est non vide. Si tel est le cas, la donnée M est délivrée à la couche applicative du site.

(ii) Chaque nœud v possédant au moins un fils v_i dans la forêt couvrante de diffusion vérifie avant de retransmettre le message à v_i que $TabG_{v[i]}\&G \neq 0$. Si tel est le cas, le message $\langle G\&TabG_{v[i]}, M\rangle$ est envoyé à v_i.

4 Conclusion

En appliquant et en combinant les résultats que nous avions obtenu précédement, nous avons proposé une solution au problème de la construction d'un schéma de multidistribution optimal qui soit multi-sources, multi-groupes, qui tolère les défaillances intermittentes du réseau, et qui est adaptatif.

Cet algorithme propose plusieurs améliorations fonctionnelles et opérationnelles par rapport aux travaux précédents. En plus de celles déjà mentionnées et relatives à l'adaptativité des applications basées sur les s-opérateurs et les r-opérateurs, on peut noter :

(i) l'envoi d'une même donnée à plusieurs groupes n'occasionne l'envoi effectif que d'un seul message, et

(ii) la réception d'un message à destination des groupes G_1 et G_2 par un site appartenant à chacun de ces deux groupes n'occasionne la réception effective que d'un seul message d'information par l'application.

Parmi les perspectives de ce travail, nous pouvons mentionner la recherche du maintient constant, y compris en cas de pannes ou en cas d'évolution des critères, d'une forêt de distribution, éventuellement temporairement non optimale. Cela permettrait d'assurer autant que faire se peut une continuité de la distribution malgré les défaillances des éléments du réseau.

References

[1] S. K. Das, A. K. Datta, and S. Tixeuil. *Self-Stabilizing Algorithms in DAG Structured Networks.* Parallel Processing Letters, Vol. 9, No. 4, pp. 563–574, Décembre 1999. Résumé dans I-SPAN'99, pp. 190–195, Juin 1999.

[2] E. Dijkstra. *Self-Stabilizing Systems in Spite of Distributed Control*. CACM 17,11 pp. 643-644, 1974.

[3] C. Diot, W. Dabbous et J. Crowcroft. *Multipoint communication: a survey of protocols, functions, and mecanisms.* IEEE Journal on Selected Areas in Communications, 15, pp. 277-290, 1997.

[4] S. Dolev, M. Gouda, M. Schneider. *Memory requirements for silent stabilization.* PODC'96, pp. 27-34, 1996.

[5] S. Dolev, T. Herman. *Superstabilizing protocols for dynamic distributed systems.* Chicago Journal of Theoretical Computer Science, décembre 1997.

[6] S. Dolev. *Self-stabilization.* The MIT Press, 2000.

[7] B. Ducourthial. *New operators for computing with associative nets*. SIROCCO'98, pp. 55-65, Carleton Scientific, Juin 1998

[8] B. Ducourthial. *Les Réseaux Associatifs* Thèse de doctorat, Université de Paris Sud, Janvier 1999.

[9] B. Ducourthial et S. Tixeuil. *Self-stabilization with r-operators*. Distributed Computing, Vol. 14, No. 3, Juillet 2001. Résumé dans OPODIS'98, pp. 99–113, Hermes, décembre 1998.

[10] B. Ducourthial et S. Tixeuil. *Self-stabilization with path algebra*. Theoretical Computer Science, à paraître. Résumé dans SIROCCO 2000, pp. 95-110, Carleton Scientific, Juin 2000.

[11] B. Ducourthial, S. Tixeuil. *Construction auto-stabilisante de forêts de diffusion optimale* Actes d'AlgoTel 2000 (INRIA) pp. 149-154. La Rochelle, France, mai 2000.

[12] M. G. Gouda and T. Herman. *Adaptive programming*. IEEE Transactions on Software Engineering, 17:911-921, 1991.

[13] M. G. Gouda and N. Multari. *Stabilizing communication protocols*. IEEE Transactions on Computers, 40:448-458, 1991.

[14] M.-G. Gouda et M. Schneider. *Maximum flow routing.* WSS'95, pp. 2.1-2.13, 1995.

[15] S. K. S. Gupta, A. Bouabdallah et P. K. Srimani. *Self-stabilizing Protocol for Shortest Path Tree for Multicast Routing in Mobile Networks*. Euro-Par'00, LNCS 1900, pp.600-604, 2000.

[16] S. K. S. Gupta et P. K. Srimani. *Using self-stabilization to design adaptive multicast protocol for mobile had hoc networks.* DIMACS Workshop on Mobile Networks and Comp., Rutgers Univ. 1999.

[17] T. Herman. *A Comprehensive Bibliography on Self-Stabilization*. A Working Paper in the Chicago Journal of Theoretical Computer Science. Disponible à `http://www.cs.uiowa.edu/ftp/selfstab/bibliography/`.

[18] M. Jayaram, G. Varghese. *Crash failures and repairs can drive protocols to arbitrary states.* PODC'96, pp. 247-256, 1996.

[19] M. Schneider. *Self-stabilization*. ACM Computing Surveys, vol. 25, pp. 45-67, 1993.

[20] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.