h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·so GENÈVE
Haute École Spécialisée
de Suisse occidentale

Systèmes Distribués

Professeur : Nabil Abdennadher

Année Universitaire : 2017/2018

Les algorithmes distribués de base

Version Octobre 2017

I. Introduction

Ce document présente les algorithmes distribués de base étudiés dans le cade du cours « Systèmes distribués » de la formation Bachelor en Ingénierie des Technologies de l'Information de hepia (HES-SO//Genève). Ces algorithmes seront étudiés sur le plan théorique : complexité, nombre de messages générés, conditions d'arrêts. Ils supposent que le réseau est fonctionnel et que les communications se font par transmission de message.

Le réseau est représenté par un graphe non orienté où chaque nœud peut être un ordinateur, un commutateur (switch), un routeur, un capteur, etc. Les liens inter-nœuds représentent les liaisons matérielles (filaire ou sans fils).

II. Construction d'un arbre de recouvrement (Spanning Tree)

Un Spanning Tree (ST, arbre de recouvrement en français) est une structure arbre qui permet le recouvrement d'un graphe. Un arbre ST recouvre un graphe G si :

- G et ST ont les mêmes nœuds et
- les arêtes de ST sont un sous ensemble des arêtes de G.

Parcourir les nœuds du graphe G revient donc à parcourir l'arbre ST. En effet, il est plus simple d'écrire des algorithmes de parcours d'arbre que des algorithmes de parcours de graphe : la structure hiérarchique d'un arbre (relation père - fils) est plus simple que la structure connexe qu'un graphe. Un arbre ST est représenté par sa racine. Chaque nœud de l'arbre connaît son père et ses fils. Dans une architecture distribuée, chaque nœud dispose d'une vue partielle de l'arbre : son père (NULL pour le nœud racine) et ses fils. Les nœuds feuilles n'ont pas de fils.

Un graphe G peut avoir plusieurs arbres ST de racines différentes. De plus, pour la même racine, un graphe G peut avoir plusieurs ST. La figure 1 montre un graphe G avec un exemple d'arbre ST. Les arêtes en rouge représentent les liens de l'arbre ST. Les arêtes en noir représentent les liens du graphe G.

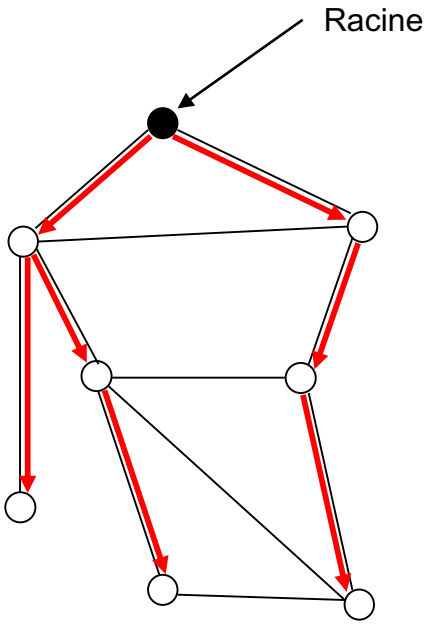

Figure 1 : Exemple d'un graphe G et son arbre ST

L'arbre ST est utilisé entre autres pour effectuer des opérations de diffusion (broadcast) et de collecte (convergecast). Les algorithmes de parcours d'arbre étant plus facile à mettre en œuvre et plus rapides (moins complexes) que les algorithmes de parcours de graphe.

Cette section détaille un algorithme distribué qui génère un arbre ST d'un graphe G donné. Chaque nœud du réseau connaît ses voisins directs, autrement dit les nœuds qui lui sont directement

connectés. La génération de l'arbre ST se fait à partir d'un nœud donné du graphe G : n_r. n_r sera alors la racine de l'arbre ST.

Trois types de messages sont échangés entre les nœuds du réseau (graphe) :

M : Ce message est une demande d'adoption. Un nœud n_i envoie un message *M* à un voisin n_j signifie que n_i demande que n_j soit son fils.

P : Ce message est envoyé par un nœud n_i à un nœud n_j lorsque n_i accepte que n_j soit son père. *P* est envoyé par n_i à n_j en réponse à l'envoi d'un message *M* par n_j à destination de n_i. Le message *M* est dans ce cas le premier à être reçu par n_i.

R : Ce message est envoyé par un nœud n_i à un nœud n_j pour lui refuser sa demande d'adoption. Ceci signifie que n_i a déjà reçu une demande d'adoption de la part d'un autre nœud n_k à qui il a déjà envoyé un message *P*.

II.1 Algorithme de construction d'un arbre de recouvrement en largeur d'abord

L'algorithme de génération de l'arbre spanning tree (ST) est donc le suivant :

Pour le noeud n_r

Envoie d'un message <M> à tous les voisins

Pour un message <M> reçu n_i envoie le message <R> à l'émetteur

Pour un message <P> reçu Actualiser la liste des processus fils : F_r

Pour un message <R> reçu Actualiser la liste des processus non fils : NF_r

Pour chaque noeud n_i (i <>r)

A la première réception d'un message <M> de la part d'un noeud n_j

n_i envoie un message <P> à n_j

$parent_i = n_j$

n_i envoie un message <M> à tous les voisins autres que n_j (I)

Pour les autres messages <M> reçus

n_i envoie le message <R> à l'émetteur

Pour un message <P> reçu

Actualiser la liste des processus fils : F_i

Pour un message <R> reçu

Actualiser la liste des processus non fils : NF_i

Alg. 1 : Algorithme de construction d'un arbre de recouvrement (Spanning Tree)

F_i (resp. NF_i) est l'ensemble des nœuds fils (resp. non fils) de n_i. Les nœuds appartenant au deux ensembles F_i et NF_i sont forcément des voisins de n_i.

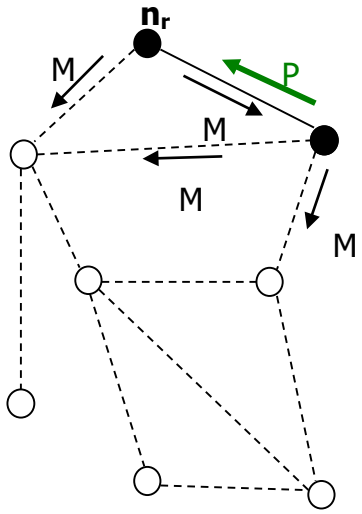

Figure 2 : Envoi d'un message <P> au premier nœud qui transmet un message <M>

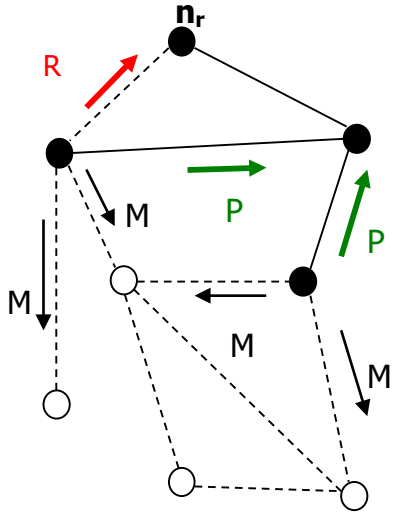

Figure 3 : Des messages <P> et <R> sont renvoyés en réponse à une demande d'adoption <M>.

A la suite de la réception du premier message *M* de la part d'un nœud n_j, un nœud n_i non racine envoie des messages <M> à tous ses voisins excepté son père n_j (Figure 2). n_i recevra donc des messages de retour de type <P> ou <R> de la part de tous ses voisins autre que le père n_j (Figure 3). Le nombre de messages reçus de type <P> et <R> est donc égal à $|v_i| - 1$. v_i est l'ensemble des voisins du nœud n_i. $|v_i|$ est la cardinalité de v_i.

L'algorithme s'arrête donc lorsque :

$|v_i| - 1 = |NF_i| + |F_i|$.

Pour le noeud racine n_r, l'algorithme s'arrête lorsque :

$|v_r| = |NF_r| + |F_r|$.

La boucle indiquant la condition d'arrêt n'est pas explicitée dans l'algorithme *Alg.1*. Cette condition est expliquée dans les slides du cours.

L'envoi simultané de plusieurs messages de type M à tous les voisions (ligne I dans l'algorithme) permet de construire des arbres « larges ». L'algorithme privilégie l'adoption de plusieurs fils par le même père. Cet algorithme est appelé : algorithme de construction d'arbre de recouvrement en largeur d'abord.

II.2 Algorithme de construction d'un arbre de recouvrement en profondeur d'abord

Une seconde version de cet algorithme permet de construire des arbres en profondeur d'abord et ce en envoyant séquentiellement les messages. L'envoie d'une nouvelle adoption ne peut avoir lieu que lors de la réception de la réponse de la demande d'adoption précédente. Une version détaillée de cet algorithme se trouve dans la présentation .ppt. (voir site web du cours).

II.3 Algorithme de construction d'un arbre de recouvrement sans racine fixée a priori

Une troisième version de cet algorithme consiste à parcourir l'arbre en profondeur d'abord. Dans cette version, la racine n'est pas connue à l'avance. Elle est déterminée au fur et à mesure de l'exécution de l'algorithme. Les nœuds qui désirent être racine lancent l'exécution de l'algorithme.

Dans cette version, Les demandes d'adoption incluent aussi l'identifiant du nœud racine qui a lancé l'exécution. A la réception d'une demande d'adoption, un nœud (n) compare l'identifiant de la racine de l'arbre auquel il appartient (leader) à l'identifiant de la demande qu'il vient juste de recevoir (y). Trois cas sont possibles :

1. Si y < leader : cela signifie que l'arbre qu'aimerait construire y doit être bloqué. Dans ce cas, le nœud n ne propage pas les demandes d'adoption. La construction de l'arbre ne sera jamais finie.
2. y = leader : cela signifie que le nœud n reçoit une demande d'adoption de y qui n'est pas la première. n envoie alors un refus de demande d'adoption.
3. y > leader : le nœud n doit changer d'arbre. Il était membre de l'arbre dont la racine est leader. Il doit maintenant migrer vers un autre arbre, celui dont la racine est y.

Les détails de cet algorithme se trouvent dans la présentation .ppt du cours.

III. Algorithmes de diffusion

Une opération de diffusion transmet la même donnée D à tous les nœuds d'un réseau. Un algorithme de diffusion transmet donc une donnée D à un ou plusieurs nœuds vers tous les nœuds du réseau.

III.1 Diffusion à partir d'un arbre de recouvrement (Spanning Tree)

Cet algorithme suppose que le Spanning Tree (ST) est déjà généré auparavant. Chaque nœud n_i détient les variables suivantes :

$parent_i$: père du nœud n_i,

F_i : ensemble des fils du nœud n_i,

L'arbre ST est représenté par la racine n_r.

Un nœud intermédiaire (différent de la racine et non feuille) reçoit la donnée D en provenance de son père et la transmet vers ses fils. Le nœud racine envoie la donnée D vers tous ses fils. Un nœud feuille ne fait que recevoir la donnée de son père.

Chaque arête de l'arbre ST transmet une seule fois la donnée D. Le nombre de messages transmis est donc égal à $n - 1$. n étant le nombre de nœuds dans le graphe G (ou l'arbre ST). En effet, le nombre d'arêtes de l'arbre ST est égal à $n - 1$.

II.2 Diffusion (broadcast) avec vague

N_0 représente l'ensemble de nœuds qui disposent de la donnée D avant l'exécution de l'algorithme de diffusion.

Le principe de cet algorithme de diffusion est le suivant : un nœud n_i n'appartenant pas à N_0 attend la réception de la donnée D. Une fois la données D reçue, n_i envoie D à tous ses voisins y compris le nœud émetteur n_j.

L'algorithme est le suivant :

$atteint_i$: booléen initialisé à faux.

Si n_i appartient à N_0

$atteint_i$ = vrai

Envoyer D à tous les voisins directs (v_i)

Finsi

A chaque réception de D

Si ($atteint_i$ =faux)

$atteint_i$:= vrai

Envoyer D à tous les voisins directs de n_i

Finsi

Finpour

Le booléen $atteint_i$ est mis à vrai si le nœud n_i appartient à N_0 ou à la réception de la donnée D pour la première fois. La donnée D est donc diffusée par « vague » (Figure 4).

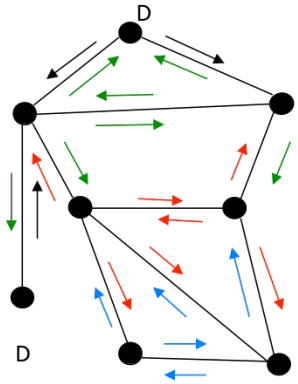

Figure 4 : Diffusion par vague. Chaque nœud crée sa propre «vague»

Chaque arête du graphe transmet la donnée D deux fois, une fois dans chaque sens. Le nombre de messages transmis est donc égal *2m*. *m* étant le nombre d'arêtes dans le graphe G.

III.3 Diffusion (broadcast) par vague avec accusé de réception

Cet algorithme est une dérivée de l'algorithme de diffusion par vague. L'ensemble N_0 est supposé être un singleton ($N_0 = \{n_r\}$). Le nœud n_r qui dispose initialement de la donnée D doit recevoir un accusé de réception indiquant que celle-ci a été transmise correctement. L'algorithme est le suivant :

$atteint_i$:= faux

$count_i = 0$

Si n_i appartient à N_0

$atteint_i$:= vrai

Envoyer D à tous les voisins directs

Finsi

Pour chaque réception de D de la part de n_j

$count_i = count_i + 1$;

Si $atteint_i$ =faux

$atteint_i$:= vrai ;

$parent_i := n_j$

Envoyer D à tous les voisins directs de n_i avec $n_i \neq n_j$

Finsi

Si $count_i = |v_i|$ *et* $parent_i \neq NULL$

Envoyer D à $parent_i$

Finsi

Finpour

Lors de la réception du premier message (donnée D) en provenance du nœud n_j, un nœud n_i initialise sa variable $parent_i$ à n_j. Le compteur $count_i$ est initialisé à 0. Il est incrémenté de 1 à chaque réception de la donnée D. L'algorithme s'arrête lorsque $count_i$ est égal au nombre de voisins, la donnée D est est alors transmise au parent du nœud n_i (ce message est considéré comme un accusé de réception).

Le nombre de messages envoyés par un nœud est égal au nombre de ses voisins. Le nombre total de messages est donc égal à *2m*. *m* étant le nombre d'arêtes dans le graphe. La complexité de cet est la même que l'algorithme « Broadcast par vague » mais son temps d'exécution est plus long puisque chaque nœud attend de recevoir tous les messages de ses voisins avant d'envoyer le dernier message au nœud qui lui envoyé en premier la donnée D.

A la fin de son exécution, l'algorithme broadcast par vague avec accusé de réception construit un arbre de recouvrement. Le père d'un nœud n est celui qui lui envoie le message la première fois. Si chaque nœud connaît sont père, il est donc possible de construire l'arbre de recouvrement.

IV. Algorithme de collecte

Un algorithme de collecte (convergecast) permet la collecte de données à partir d'un ou plusieurs noeud du graphe. Chaque nœud dispose d'une donnée locale D_i. A la fin de l'exécution d'un algorithme de collecte, toutes les données Di sont stockées chez le nœud n_r. On suppose qu'un arbre ST, dont la racine est n_r, est d'ores et déjà connu. Chaque nœud de l'arbre connaît ses fils et son père. Rappelons que le nœud racine n'a pas de père et les nœuds fils n'ont pas de fils. Les nœuds feuille commencent à envoyer leurs données à leur père. A la réception des toutes les données à partir de ses fils, un nœud intermédiaire envoie l'ensemble des données reçues à son père. Enfin, le nœud racine se bloque jusqu'à la réception de toutes les données à partir de ses nœuds fils. L'algorithme est le suivant :

Chaque noeud feuille

Envoi la donnée locale D_i au noeud père

Chaque noeud non feuille n_i ***(i ≠ r)***

A la réception des messages de tous les noeuds fils

Envoi l'ensemble des messages reçus au noeud père

noeud n_r

Reçoit les messages de la part de tous les noeuds fils

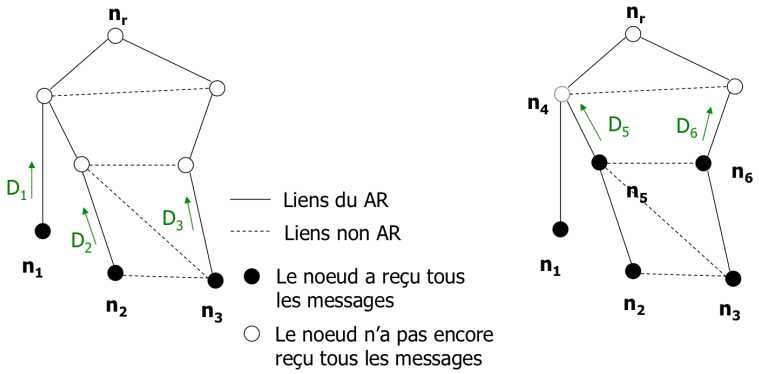

Figure 5 : Etape 1 : Transmission de D_1, D_2, D_3 par les nœuds feuille n_1, n_2, n_3.

Etape 2 : Transmission de D_5, D_6 par les nœuds intermédiaires n_5, n_6.

V. Calcul des plus courts chemins

Ce paragraphe propose un algorithme distribué qui calcule le plus court chemin (exprimé en terme de nombre de liens) entre deux nœuds quelconque du réseau (graphe). A la fin de l'exécution de l'algorithme, chaque nœud n_i est informé :

- de la plus petite distance qui le sépare de chacun des autres nœuds
- des plus courts chemins qui relient n_i aux autres nœuds du graphe. Pour chacun de ces chemins, l'information se trouve distribuée entre les nœuds appartenant au chemin en question. En supposant qu'il s'agit du chemin reliant n_i à n_j, n_i connaît le voisin direct n_k qui appartient à ce chemin. n_k connaît à son tour le prochain nœud voisin direct n_h qui le relie au même nœud n_j, et ainsi de suite.

Les plus courtes distances séparant deux nœuds quelconque du graphe sont utilisées entre autres lors du routage des messages de données.

Le paragraphe V.1 décrit un algorithme synchrone qui calcule les plus courts chemins dans un graphe de nœuds donné.

V.1 Algorithme synchrone

Dans ce qui suit, on notera par ***dist$_i$ (j)*** la plus petite distance qui sépare le nœud n_i du nœud n_j. L'algorithme s'exécute en plusieurs étapes. Une étape *t* ne s'exécute qu'après la fin de l'étape *t–1*.

A l'instant *t = 0*, chaque nœud n_i envoie son identificateur à tous ses voisins. A l'instant *t = 1*, chaque nœud n_i dispose donc des identificateurs des nœuds n_j tels que *dist$_i$ (j)* = 0 ou 1 (le nœud lui-même et ses voisins directs). n_i peut alors construire l'ensemble des nœuds n_j tels que *dist$_i$ (j)= 1* et les transmets vers ses voisins directs. A l'instant t = 2, chaque nœud reçoit alors les identifiants des nœuds qui sont distants de ses voisins d'une distance de 1, il peut alors construire l'ensemble des nœuds n_j tels que *dist$_i$(j)= 0, 1 ou 2* (le nœud lui-même, ses voisins et les voisins de ses voisins). Comme le nœud n_i connaît les nœuds qui sont distants de lui de 0 et 1, il peut alors construire l'ensemble des nœuds qui vérifient la condition : *dist$_i$(j)=2* avant de les envoyer à ces voisins directs.

De manière générale, à l'instant *t >= 0*, chaque nœud n_i envoie à ses voisins l'ensemble des nœuds n_j tels que la distance *dist$_i$ (j) = t*. Pour *t = 0*, cet ensemble est le singleton n_i. Pour *t > 0*, cet ensemble représente tous les nœuds reçus durant les précédents *t–1* instants et qui ont une distance de n_i au plus égale à *t* : Ils ont une distance *t-1* des nœuds voisins de n_i auquel on ajoute 1).

Le nombre d'impulsions (instants) maximum pour l'exécution de cet algorithme est égal à *n* (nombre de nœuds). Le nombre effectif d'impulsions varie d'un nœud à un autre. Un nœud n_i s'arrête d'envoyer des messages à un instant *t* lorsque l'ensemble des nœuds générés par n_i à cet instant est vide. Toutefois, le même nœud peut continuer à recevoir des messages de la part de ses voisins.

En plus des variables *dist$_i$ (j)*, définies au niveau de chaque nœud *i*, l'algorithme synchrone qui calcule les plus courts chemin (appelé dans ce qui suit *S_Calcul_PCC*) entre deux noeuds quelconque du réseau, utilise les variables suivantes :

first$_i$(k) : nœud de v_i qui appartient au plus court chemin entre les nœuds n_i et n_k. Initialement, *first$_i$(k)= nil.*

set$_i$(t) : Ensemble des nœuds à envoyer aux voisins de n_i à l'instant *t*. Initialement, *set$_i$= {n_i}*.

MSG$_i$ (t) = Ensemble complet des identifiants des nœuds reçus par le nœud n_i à l'instant *t*.

Dans cet algorithme, tous les nœuds lancent instantanément l'exécution de l'algorithme S_Calcul_PCC. N_0 est donc égal à *N*.

Algortihme *S_Calcul_PCC* (s'exécutant sur le nœud n_i)

Variables

dist$_i$(i) = 0 ;

dist$_i$(k) = ¦N¦ pour tous les noeuds de N tels que k ≠ i ;

first$_i$(k) = nil ;

Début

t = 0 ; MSG$_i$ (0) = {} ;

Pour (tous les nœuds de N_0) faire envoyer *set$_i$(t)* aux voisins (ensemble v_i)

Pour (chaque réception d'un *MSG$_i$(t)*, 0 < t <= ¦N¦ - 1) faire

set$_i$(t) = {} ;

Pour (chaque $set_j(t)$ appartenant à $MSG_i(t)$) faire

Pour chaque nœud n_k appartenant à $set_j(t)$ faire

Si ($dist_i(k) > t$) alors

$dist_i(k) = t$;

$first_i(k) = n_j$;

$set_i = set_i + \{ n_k \}$

Envoyer set_i à tous les voisins de n_i (ensemble v_i)

Fin

Selon cet algorithme, chaque nœud reçoit l'identification de tous les autres nœuds du réseau. Chaque identification traverse tous les liens du réseau une fois dans chaque sens. Le nombre de messages transmis est donc égal à 2*n*m (n étant le nombre de nœuds et m est le nombre de liens).