UPMC
PARIS UNIVERSITAS

UFR 919

février 2010

Master d'Informatique 2009 -2010

spécialité « SAR »

Algorithmique Répartie (MI048)

Cours 6 à 10

L. Arantès, C. Dutheillet, M. Potop-Butucaru, S. Dubois

Systèmes Répartis
- / -
Etat global

Plan

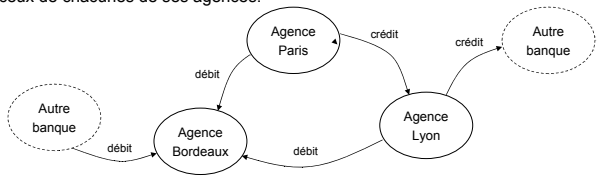
Plan (2)

I. Introduction

I.1 Motivation

Le calcul de l'état global d'un système réparti consiste à prendre un instantané de l'état du système à un moment donné de son exécution. Plusieurs utilisations possibles :

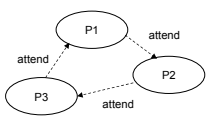
1- **Exécution d'un algorithme centralisé** sur l'état global, pour s'affranchir du caractère réparti du système. Exemple : calcul des avoirs d'une banque à partir de ceux de chacunes de ses agences.



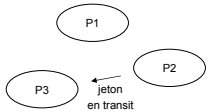
$Avoirs_{banque} = Avoirs_{agence\ Paris} + Avoirs_{agence\ Lyon} + Avoirs_{agence\ Bordeaux}$

I.1 Motivation (2)

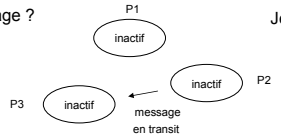
2- **Détection d'états globaux stables**, tels que l'inter-blocage ou la terminaison d'un algorithme distribué.



Inter-blocage ?



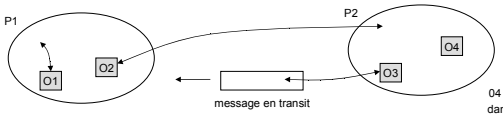
Jeton perdu ?



Terminaison ?

I.1 Motivation (3)

3- **Ramasse-miette distribué** (élimination des objets qui ne sont plus référencés par aucun des processus du système réparti).



04 n'est plus référencé dans le système réparti

03 n'est plus référencé pour l'instant que dans le message en transit

4- **Tolérance aux fautes** : l'état global du système est sauvegardé pour servir de point de reprise. En cas de panne d'un des sites, le système réparti est replacé dans le dernier état global sauvegardé, et relancé à partir de cet état.

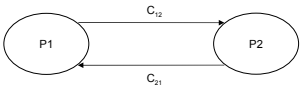
I.2 Définitions

Etat local d'un processus P_i : valeur des variables locales de P_i (et plus généralement de son contexte d'exécution).

Note : si P_i est un processus déterministe, l'état local de P_i est déterminé par son état initial et la succession des événements, notamment les événements réception s'étant produit sur P_i.

Etat du canal de communication C_{ij} entre les processus P_i et P_j : ensemble (ordonné, si le canal est FIFO) des messages en transit entre P_i et P_j.

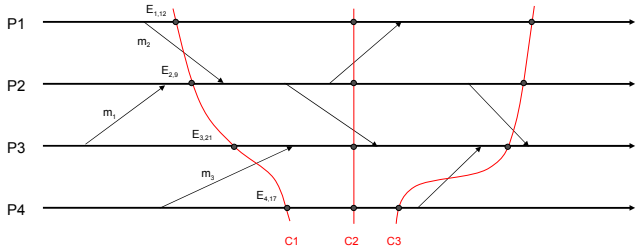
Note : on considère ici que les canaux de communication sont unidirectionnels ; il y a deux canaux de communication entre les processus P_i et P_j : C_{ij}, de P_i vers P_j, et C_{ji}, de P_j vers P_i.



Etat global d'un système réparti : union de l'ensemble des états locaux des processus P_i et de l'ensemble des états des canaux C_{ij} constituant le système réparti.

I.2 Définitions (2)

Un état global peut être représenté par une courbe, appelée aussi **coupure**, sur le diagramme temporel du système réparti. La coupure divise le diagramme en deux zones (sur chaque processus) : passé et futur.

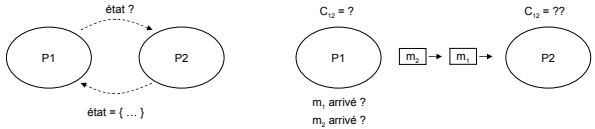


$E_{C1}(SR) = E(P_1) = E_{1,12} \cup E(P_2) = E_{2,9} \cup E(P_3) = E_{3,21} \cup E(P_4) = E_{4,17} \cup E(C_{12}) = \{m_2\} \cup E(C_{43}) = \{m_3\}$

I.3 Problèmes

Problème de la **collecte** des états :

- $E(P_i)$ n'est directement et immédiatement observable que sur P_i
- $E(C_{ij})$ n'est jamais directement observable, ni sur P_i, ni sur P_j



Problème de la **cohérence** des états collectés :

Idéalement, il faudrait figer l'état des différents éléments (processus et canaux) au même instant absolu (exemple : coupure C2) pour qu'ils soient tous en cohérence.

Mais il n'y a pas de référence temporelle commune : il faut définir un critère de cohérence plus souple, mais qui donne néanmoins un état global exploitable.

I.4 Coupure cohérente

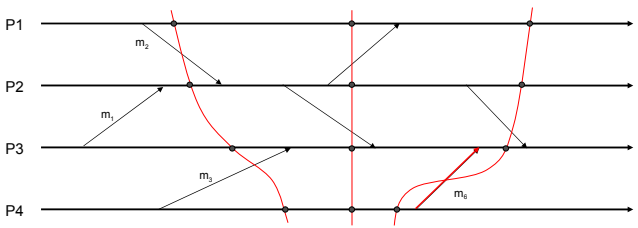
Une coupure est **cohérente** si elle définit un passé fermé pour la relation de précédence causale. Tous les événements ayant potentiellement causé un événement quelconque se situant avant la coupure sont également avant la coupure. Formellement :

C coupure cohérente $\Leftrightarrow \forall e \in Passé(C), e' \rightarrow e \Rightarrow e' \in Passé(C)$

Intuitivement : la cause d'un événement ne peut être dans le futur (de la coupure). C'est la condition minimale pour qu'une coupure puisse correspondre à un état global réellement atteint lors de l'exécution du système réparti (sans que cet état global ait *nécessairement* été atteint).

Par définition, un état **global cohérent** est un état global obtenu selon une coupure cohrénte.

I.4 Coupure cohérente (2)



C1 et C2 sont cohérentes

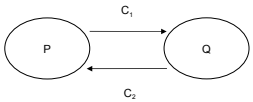
C3 **n'est pas** cohérente : réception(m_6) ∈ Passé(C3) et émission(m_6) ∉ Passé(C3)

Pour déterminer si une coupure est cohérente, il suffit d'examiner les messages en transit : aucun message ne doit aller du futur vers le passé.

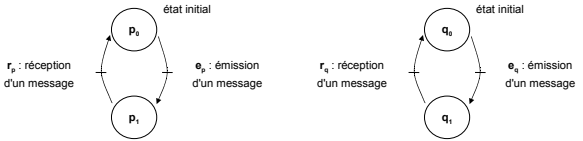
II. Exemple d'état global

II.1 Définition du système étudié

Système réparti étudié : deux processus P et Q, communiquant par deux canaux unidirectionnels C_1 et C_2.



Chacun des processus P et Q envoie puis recoit un message, alternativement. Automates des processus :



Etat global du système : quadruplet <état local P, état C1, état C2, état local Q>

II.2 Graphe des états accessibles

Graphe des états accessibles : c'est l'ensemble des états que peut prendre le système réparti, à partir de son état initial.

Un **nœud** du graphe correspond à un état du système réparti :

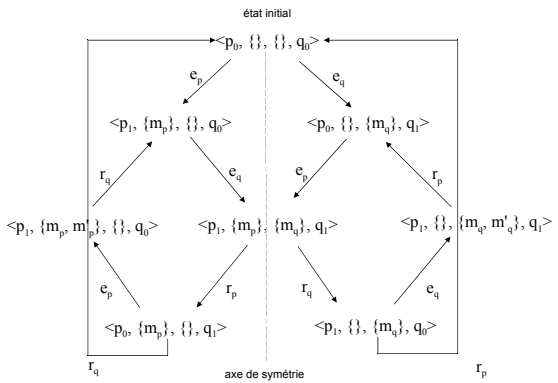
quadruplet <état local P, état C1, état C2, état local Q>

Une **arrête** correspond à une transition d'un état à un autre suite à un événement :

e_p, r_p, e_q ou r_q

Une exécution particulière du système réparti correspond à **un** cheminement dans ce graphe à partir de l'état initial.

II.2 Graphe des états accessibles (2)



II.3 Remarques

La connaissance du graphe des états accessibles permet de déduire plusieurs propriétés intéressantes du système :

1- il n'existe aucun état puit (sans arrête sortante) dans le graphe : cela traduit l'absence de deadlock pour toute exécution du système réparti.

2- on constate que pour toute exécution, il y a au plus deux messages dans les canaux C1 et C2 : cela permet de dimensionner ces canaux à priori.

Autres remarques :

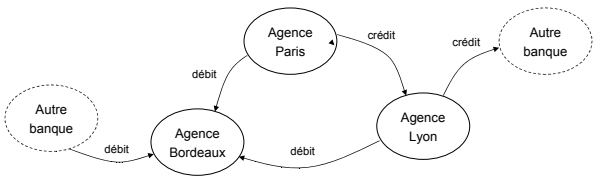
- chacun des états du graphe est cohérent, puisqu'il correspond à un état réellement accessible.
- le système réparti est symétrique : le graphe est également symétrique.
- bien que le système soit simple, le graphe comprend déjà 8 états.

III. Algorithme de snapshot de Chandy et Lamport 85

III.5 Objectif

Il s'agit de calculer **dynamiquement** un **état global cohérent** d'un système réparti en cours d'exécution :

- de façon simple,
- si possible sans interférer avec le fonctionnement normal du système.



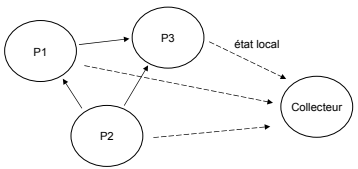
$Avoirs_{banque} = Avoirs_{agence\ Paris} + Avoirs_{agence\ Lyon} + Avoirs_{agence\ Bordeaux}$?

III.2 Hypothèses

Les processus ne tombent **pas en panne** (durant le snapshot).

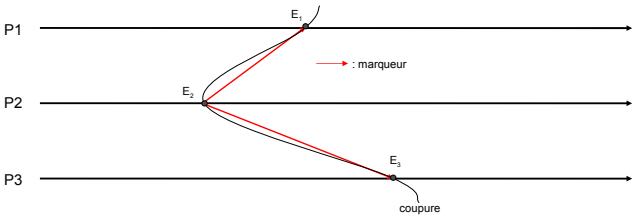
Le réseau d'interconnexion est fortement connexe et les canaux de communication sont **fiables** et **FIFO**.

Un processus **collecteur** est désigné pour collecter l'état global, ce processus est connu de tous les autres (c'est éventuellement l'un d'entre eux).



III.3 Analyse

On part de l'algorithme le plus simple pour lister les problèmes à résoudre. Un processus déclenche le calcul d'un état global : il sauvegarde sont état local et demande aux autres processus d'en faire autant en leur envoyant un message spécial (marqueur).

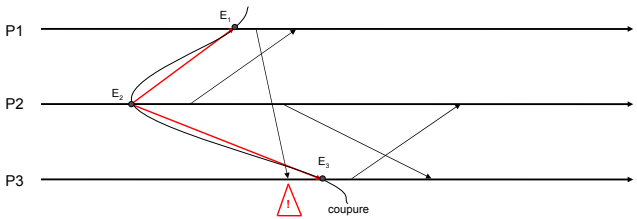


On doit assurer deux propriétés : la **cohérence** de la coupure, et la **complétude** l'état sauvegardé pour les canaux de communication.

III.3 Analyse (2)

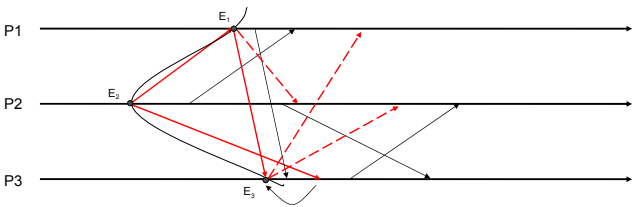
Cohérence : un message émis **après** une sauvegarde doit être reçu **après** la sauvegarde correspondante du processus destinataire.

- de P_2 vers P_1 et P_3 : pas de problème puisque les canaux sont FIFO
- de P_3 ou P_1 vers P_2 : pas de problème en raison de la causalité
- aucune garantie pour les autres couples de processus



III.3 Analyse (3)

Solution pour la cohérence : propager le marqueur. Chaque processus faisant une sauvegarde (ré)-émet le marqueur vers tous les autres. Chaque processus fait une sauvegarde sur réception du premier marqueur, mais pas sur les suivants.



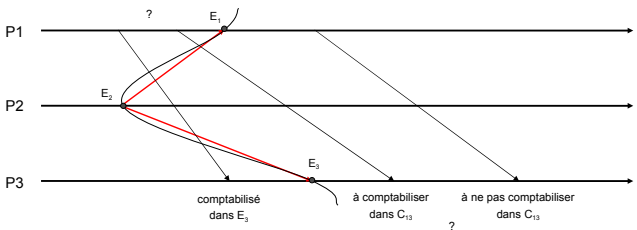
La sauvegarde E_3 a été avancée, de sorte que le message de P_1 vers P_3 qui posait problème arrive maintenant après la sauvegarde.

III.3 Analyse (4)

Complétude : un messages émis **avant** une sauvegarde doit être comptabilisé :

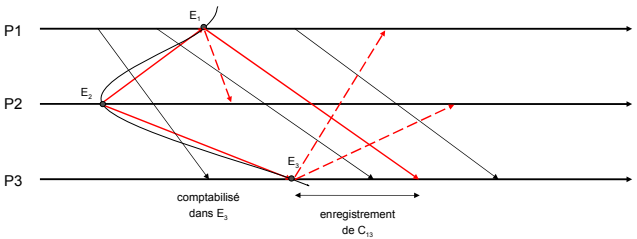
- soit dans la sauvegarde du destinataire en étant reçu **avant** cette sauvegarde,
- soit dans l'état du canal parce qu'il a été reçu après cette sauvegarde.

Problème : quand débuter et arrêter l'enregistrement des messages en transit ? Quels sont les messages émis qui ont été reçus avant la sauvegarde du destinataire ?



III.3 Analyse (5)

Solution pour la complétude : partir non des messages émis, mais des messages reçus, et utiliser là aussi les marqueurs propagés. Les messages en transit sont ceux arrivés après la sauvegarde et avant le renvoi du marqueur par les processus émetteurs.



Note : C_{23} est par définition vide car P_3 reçoit son premier marqueur de P_2.

III.4 Principes

Utilisation de messages **marqueurs** (messages de service, en plus des messages applicatifs normaux).

Mise en oeuvre de deux règles :

- règle d'envoi du marqueur : quand un processus sauvegarde son état, il envoie un marqueur à tous les autres, **avant** tout message applicatif
- règle de réception du marqueur : sur réception du premier marqueur, un processus sauvegarde son état, et débute l'enregistrement des messages reçus sur chacun de ses canaux, enregistrement qui se termine sur réception du marqueur associé au canal

Un processus souhaitant débuter la collecte d'un état global agit comme sur réception d'un marqueur (fictif).

III.5 Algorithme

Variables locales :
enreg$_i$ = faux;
etatLocal$_i$ = ∅;
etatCanal[N]$_i$ = {∅, ∅, …};
marqRecu[N]$_i$ = {faux, faux, …};

sauvegarder() :
enreg$_i$ = vrai;
etatLocal$_i$ = etatLocal();
POUR j = 1 A N SAUF i
  envoyer(MARQ, P_j)
  etatCanal[j]$_i$ = ∅;
  marqRecu[j]$_i$ = faux;
FPOUR

recevoir(m, P_j **)** :
SI m == MARQ ALORS
  SI ! enreg$_i$ ALORS
    sauvegarder()
  FSI
  marqRecu[j]$_i$ = vrai;
  SI toutRecu() ALORS
    envoyer(<etatLocal$_i$, etatCanal[]$_i$, Collecteur);
    enreg$_i$ = faux;
    marqRecu[]$_i$ = {faux, faux, …};
  FSI
  RETOURNER
FSI
SI enreg$_i$ ET ! marqRecu[j]$_i$ ALORS
  etatCanal[j]$_i$ ∪= { m };
FSI

III.5 Algorithme (2)

etatLoca() :
RETOURNER variables locales du processus

toutRecu() :
POUR j = 1 A N SAUF i
  SI ! marqRecu[j]$_i$ ALORS
    RETOURNER faux;
  FSI
FPOUR
RETOURNER vrai;

III.7 Propriétés

L'algorithme se termine.

L'état global enregistré est complet.

L'état global enregistré est cohérent.

Remarque :
L'état enregistré peut ne pas correspondre à un état global effectivement atteint par le système au cours de la même exécution. Mais on peut montrer qu'il correspond à un état global que le système aurait atteint en réordonnant les événements de façon compatible avec la causalité (i.e. en réordonnant les seuls événements concurrents)

Bibliographie

V. Garg Elements of distributed computing

G. Tel Introduction to distributed algorithms

P. Sens Systèmes répartis (manuscrit)

Jean-Michel Busca – Systèmes répartis (transparents de cours)

Election de chef

Maria Gradinariu Potop-Butucaru
Université Paris 6

Election d'un chef

- Étant donné un ensemble de processus choisir un unique chef (sûreté) en un temps fini (vivacité)
- Applications :
  - Recréation d'un jeton perdu : uniquement le chef aura le droit d'introduire un nouveau jeton
  - Construction d'arbre couvrant : le chef devient la racine de l'arbre et peut initier cette construction par diffusion
  - Dans les systèmes « maître - esclave », en cas de la défaillance du maître élire un nouveau maître

Construction d'un arbre couvrant (diffusion)

- Un unique arbre couvrant est construit (sûreté) en un temps fini (vivacité)
- Utilisation d'un chef
  - Le chef commence la construction de l'arbre couvrant en envoyant un message spécifique M à ses voisins; un processus qui reçoit le message M prend comme père l'expéditeur du message et le diffuse à son tour

Construction d'un arbre couvrant (diffusion)

- Structures :
  - Parent : pointer vers le père du nœud dans l'arbre couvrant (initialement NULL)
  - Children : ensemble des fils (initialement vide)
  - Others : les voisins qui ne sont pas de fils (initialement vide)

Construction d'un arbre couvrant (diffusion)

Site i ne reçoit pas de message

```
if chef and parent=NULL then
    envoyer M aux voisins; parent=i
```

Site i reçoit message M depuis site j

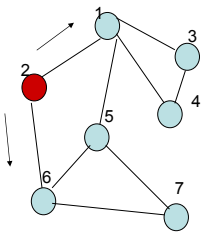
```
if parent=NULL then
  parent=j
  envoyer <parent> à j
  envoyer M aux voisins k, tels que k≠j
else envoyer <rejet> to j
```

Site i reçoit <parent> depuis site j

```
children=children U {j}
if children U others=voisins \ {parent} then
  fin
```

Site i reçoit <rejet> depuis site j

```
others=others U {j}
if children U others=voisins \ {parent} then
  fin
```



Construction d'un arbre couvrant (diffusion)

Site i ne reçoit pas de message

```
if chef and parent=NULL then
    envoyer M aux voisins; parent=i
```

Site i reçoit message M depuis site j

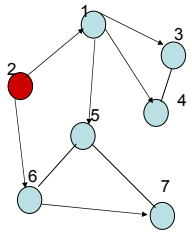
```
if parent=NULL then
  parent=j
  envoyer <parent> à j
  envoyer M aux voisins k, tels que k≠j
else envoyer <rejet> to j
```

Site i reçoit <parent> depuis site j

```
children=children U {j}
if children U others=voisins \ {parent} then
  fin
```

Site i reçoit <rejet> depuis site j

```
others=others U {j}
if children U others=voisins \ {parent} then
  fin
```



Others(6)={5}
Others(5)={7,6}
Others(3)={4}
Others(4)={3}
Others(7)={5}

Election d'un chef

- Il n'existe pas d'algorithme déterministe d'élection de chef dans les réseaux anonymes et uniformes
- Idée de la preuve :
  - Le réseau est anonyme donc la configuration de départ peut être symétrique
  - La configuration objectif (celle où un leader est élu) est une configuration asymétrique
  - Il existe une exécution du système telle qu'à partir d'une configuration symétrique on passe toujours dans une configuration symétrique

Election d'un chef

- Contourner les résultats d'impossibilité :
  - via les identifiants
    - Chang et Roberts (anneau)
    - Hirshberg-Sinclair
    - diffusion
  - via les algorithmes probabilistes
    - Itai et Rodeh

Election d'un chef : Chang et Roberts

- topologie : anneau unidirectionnel (chaque site i dispose d'un pointeur vers son successeur succ[i])
- plusieurs candidats simultanés possibles
- Idée : chaque candidat diffuse autour de l'anneau sa candidature; le processus ayant l'identifiant max gagne

Election d'un chef : Chang et Roberts

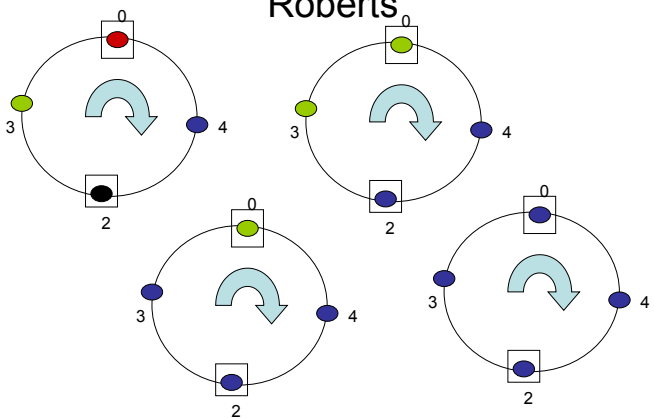
Candidature site i

```
candidat_i = vrai
envoyer(CHEF,i) à succ[i] /* i diffuse sa candidature
```

Réception sur site i du message (CHEF,j) depuis site j

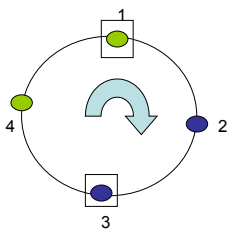
```
Case
  j>i: envoyer (CHEF,j) à succ[i]
  j<i: if not candidat_i
         candidat_i=vrai; envoyer(CHEF,i) à succ[i]
  j=i: diffusion(CHEF, i) /* envoyer à tous le résultat de l'élection
```

Election d'une chef – Chang & Roberts



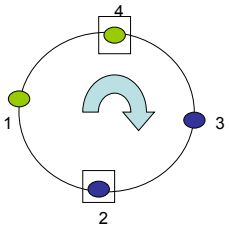
Complexité Chang et Roberts

- Le meilleur cas : O(n)
  - Les identifiants sont ordonnés dans l'ordre croissant autour de l'anneau



Complexité Chang et Roberts

- Le pire cas : $O(n^2)$
  - Les identifiants sont ordonnés dans l'ordre decroissant autour de l'anneau
  - L'identifiant « i » visite i noeuds avant de décider son status



Complexité Chang et Roberts

- En moyenne : O(nlog n)
  - Répertorier toutes les possibilités d'arranger les identifiants autour de l'anneau
  - (n-1)! (sans les configurations isomorphiques)
  - Variable aléatoire X_k: Nombre de messages si l'eléction était partie du noeud k
    - E[∑X_k] = ∑ E[X_k] pour k de 1 à n

Election d'un chef via diffusion (1)

- Idée :
  - chaque candidat envoie son identité aux autres nœuds du réseau
  - un site répond à ceux de numéro inférieur au sien
  - Un processus qui ne reçoit pas de réponse est le chef
- Hypothèse : communication fiable et synchrone (borne connue sur le temps de communication)

Election d'un chef via diffusion (2)

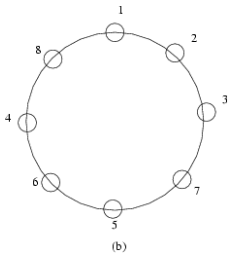
- Idée :
  - chaque candidat envoie son identité aux autres nœuds du réseau et attend les indentités des autres sites
  - Calcul du max/min sur l'ensemble d'indentités recus
- Hypothèse : communication fiable et connaissance du nombre de processus dans le réseau

Hirschberg-Sinclair

- Topologie : anneau bidirectionnel
- L'algorithme travaille en roundes
- Uniquement les processus qui gagnent l'élection du rounde *r participent au rounde r+1*
- Algorithm: P_i est le leader dans le rounde *r* ssi P_i est l'identifiant maximal dans l'ensemble de noeuds à distance au plus 2^r de P_i

Hirschberg-Sinclair

- Initialement:
  - Tous les processus sont chefs
- Rounde 0:
  - 6 , 7 et 8 sont chefs
- Round 1:
  - 7, 8 sont chefs
- Round 2:
  - 8 est le seul chef
  - au plus log(N) roundes



Itai-Rodeh

- Basé sur l'algorithme de Chang et Roberts
- Chaque processus choisit aléatoirement un idéntifiant dans l'ensemble 1..n (deux processus peuvent choisir le même identifiant)
- Chaque processus candidat envoie un jeton avec deux champs :
  - "counter" initialisé à 1
  - "another" initialisé à faux (dès que le jeton rencontre un candidat avec le même identifiant, "another" passe à vrai)
- Les leaders de la rounde "i" recommencent l'algorithme
- L'algorithme se termine avec probabilité 1

Election de chef - Applications

- Calcul de la taille d'un réseau
- Mettre en place d'un système de type publish/subscribe
- Implémenter l'allocation de ressources en exclusion mutuelle
- Accès aux données répliquées
- Implémenter le consensus
- Détecter la terminaison d'un algorithme
- Sortir des situations de blockage

Election de leader

- Réseaux de robots
  - Solutions probabilistes
- Réseaux de capteurs
  - Mise en place des algorithmes locaux probabilistes
- Réseaux P2P
  - Difficile de choisir un leader car il peut à tout moment quiter le système (ici des solutions alternatives s'imposent)

Détection Répartie de la Terminaison

Plan

- **Définition du Problème**
  - Exemple de mauvais algorithme
- **Exemple d'algorithmes**
  - Algorithme de Misra [1983]
  - Modèle à communication instantanée
    - Algorithme de Rana[1983]
    - Algorithme de Dijkstra [1983]
  - Modèle atomique :
    - Algorithme des quatre compteurs (Mattern [1987])

Détection Répartie de la Terminaison

- **Construction d'une couche de contrôle afin de détecter la terminaison d'une application répartie.**
  - Distinguer l'algorithme de détection de terminaison de l'algorithme de l'application.
    - Pas d'influence dans l'exécution de l'application
- **Configuration terminale**
  - aucune action supplémentaire de l'application ne peut être exécutée
  - Tous les canaux de communication sont vides

Détection Répartie de la Terminaison

- **État**
  - ***actif*** : si une action interne ou l'action *émettre()* est applicable
  - ***passif***
    - Dans le cas contraire
- **Message**
  - ***Applicatif ("basic message"):***
    - Message de l'application
  - ***Contrôle***
    - Message de l'algorithme de détection de la terminaison.

Détection Répartie de la Terminaison

- **Un modèle est défini pour une exécution répartie en définissant les actions des processus *actifs* et *passifs*.**
- **Les processus suivent les règles suivantes:**
  1. Initialement, chaque processus ***p*** peut être dans l'état ***actif*** ou ***passif***
  2. Un processus ***p*** peut passer spontanément de l'état ***actif*** à ***passif***.
  3. Seuls les processus ***actifs*** peuvent envoyer des massages applicatifs.
  4. Lors de la réception d'un message applicatif, un processus ***p passif*** passe à ***actif***.
     - Seule façon pour un processus ***passif*** de passer à ***actif***.
  - Observations :
    - Un message de contrôle ***émis*** lorsque le processus est *passif* ne le rend pas *actif*.
    - La ***réception*** d'un message de contrôle par un processus *passif* ne le rend pas *actif*.

Détection Répartie de la Terminaison

- **Terminaison**
  - Π : ensemble de processus
  - C : ensemble de canaux
  - Prédicat ***TERM*** :
    - **TERM <=> (∀ p _ Π : p passif) et (∀ c _ C: c vide)**
      - ***TERM*** est un prédicat stable :
        - TERM (t) = true => ∀ t' > t : TERM (t') = true

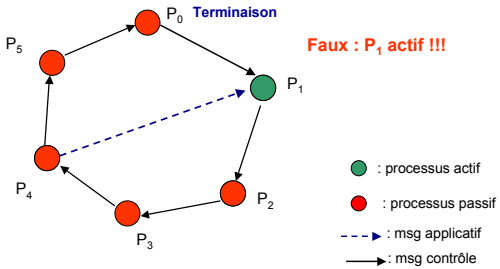
Détection Répartie de la Terminaison

- **Propriétés :**
  - ***Sûreté*** *:*
    - Si un processus détecte la terminaison à l'instant *t*, alors *TERM(t) = true*
      - Pas de fausse détection
  - ***Vivacité*** *:*
    - Si à un instant *t*, *TERM(t) = true*, alors l'algorithme de détection finira par détecter cette terminaison.

Détection Répartie de la Terminaison

- **Exemple d'un mauvais algorithme de détection répartie de la terminaison**
  - Les sites se trouvent soit dans l'état ***passif*** soit dans l'état ***actif***
  - Algorithme :
    - Faire circuler un jeton (message de contrôle) selon une structure d'anneau, envoyé initialement par P_0.
    - Lorsqu'un site est *passif* et possède le jeton, il l'envoie au site suivant.
    - Lorsque le jeton revient à P_0, la terminaison est détectée.

Mauvais algorithme de détection de la terminaison



Terminaison sur un anneau

- **Algorithme de Misra**
  - Anneau logique
    - Canaux FIFO unidirectionnels.
  - Chaque site une couleur *noir* ou *blanc.*
    - *noir =actif*
    - *blanc = passif*
  - Jeton porte un compteur
    - Nombre de sites trouvés *passif* par le jeton.
  - Terminaison détectée : tous les sites sont blancs après un tour.

Algorithme de Misra

N sites

init:
state =actif
color = black
if (i==0)
token = true
else
token = false;

Upon fin:
state =passif

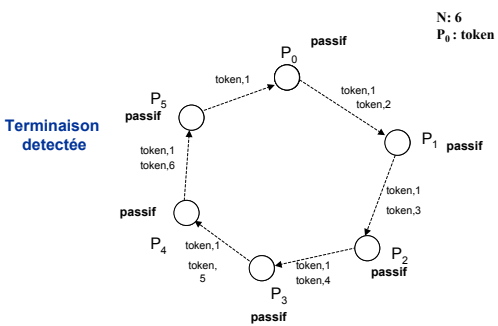
Upon reception application msg:
etat =actif
color = black

Upon reception TOKEN (count)
token = true;
Nb=count;
if ((Nb== N) and (color== white))
termination detection;

Upon (token== true) **and** (state==passif)
if (color == white)
send (TOKEN, Nb+1)
else
send (TOKEN,1):
color = white;
token = false;

Si quand on reçoit le jeton on est actif, on remet le compteur du jeton à 0, sinon on incrémente.

Algorithme de Misra



Détection Répartie de la Terminaison

- **Modèles afin de simplifier le problème :**
  - ***A communication instantanée :***
    - Communication synchrone : exemple CSP
    - **TERM <=> (∀ p _ Π: p passif)**
  - ***Atomique*** :
    - Le moment d'activité des processus est négligeable.
      - **TERM <=> (∀ c _ C: c vide)**

Modèle à communication instantanée

- **Algorithme de Rana [1983]**
  - Communication instantanée (e.g. CSP)
  - *N* sites organisés dans un anneau logique unidirectionnel.
    - Messages transmis sur l'anneau.
  - A chaque fois qu'un processus reçoit soit un message *applicatif* soit un message de *contrôle*, il met son ***horloge logique locale*** à jour.
  - Les messages de contrôles circulent sur l'anneau.
    - Message de contrôle: *<H, compteur>*
    - Chaque site envoie le message de contrôle à son successeur et le reçoit de son prédécesseur;
  - ***Observation*** : Huang [1988] a étendu l'algorithme de Rana
    - TD terminaison

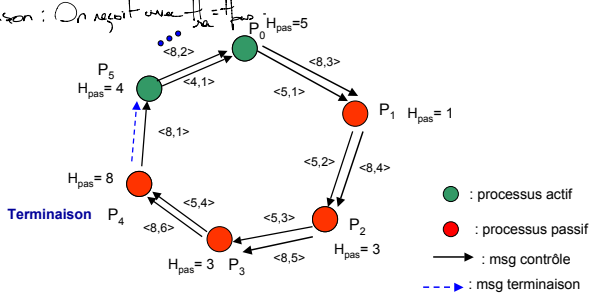
Algorithme de Rana

- **Lorsqu'un processus devient passif, il enregistre la valeur de son horloge locale(H_{pas}) et envoie le message de contrôle < H_{pas},*1*> à son successeur;**
- **Lors de la réception d'un message de contrôle :**
  - Si le site est actif, il ignore le message;
  - Sinon
    - Si (compteur !=N)
      - Si la valeur de son passage à passif H_{pas} > H_{msg} du message de contrôle reçu, le message est ignoré;
      - Sinon, le message est envoyé à son successeur avec le compteur incrémenté < H_{pas},*compteur+1*>;
    - Sinon
      - Terminaison détectée.
      - Le site envoie à son successeur un message de terminaison; Le message fera le tour de l'anneau.

Lorsque qu'on passe à passif --> envoi de <H, 1> suivant
Reception : Si H_{pas} > H_{rec} -> ∅ envoi

Algorithme de Rana

Sinon : envoi de <H_{rec}, cpt+1> au suivant
Terminaison : On reçoit une H_{rec} = H_{pas}



Modèle à communication instantanée

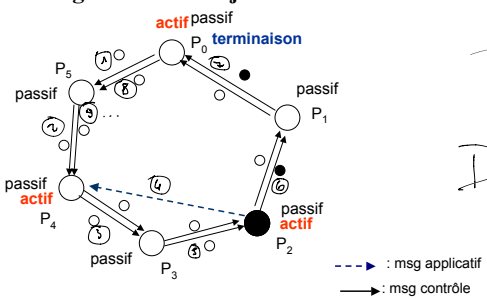
- Algorithme de **Dijkstra** [1983]
  - Modèle à communication instantanée
  - N sites organisés dans un anneau logique.
  - Existence d'un jeton
  - Les sites peuvent être de couleur blanche ou noire ainsi que le jeton.
    - Initialement tous les sites et le jeton sont blancs.

Algorithme de Dijkstra

- **Il y a un site initiateur P_0.**
  - Quand P_0 devient passif, il envoie le jeton couleur blanche à P_{N-1}.
- **Lorsque le site P_i, qui détient le jeton, devient *passif*, P_i envoie le jeton au site P_{i-1} :**
  - Si P_i est blanc :
    - P_i envoie à P_{i-1} le jeton sans changer la couleur du jeton ;
  - Sinon,
    - P_i change la couleur du jeton à noire avant de l'envoyer à P_{i-1}.
    - P_i devient blanc ;
- **Un site P_i devient noire en envoyant un message *applicatif* au site P_j.**
- **Lorsque P_0 reçoit le jeton :**
  - Si le ***jeton*** est ***blanc*** et P_0 est ***blanc*** et dans l'état ***passif***
    - **terminaison détectée**
  - Sinon
    - lorsque P_0 devient ***passif***, il renvoie le jeton couleur blanche à P_{N-1}.

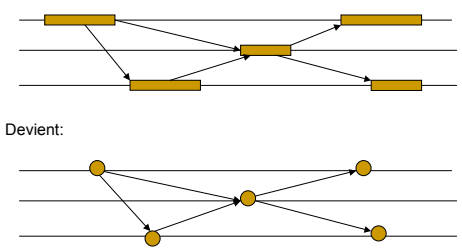
Détection Répartie de la Terminaison

Algorithme de Dijkstra



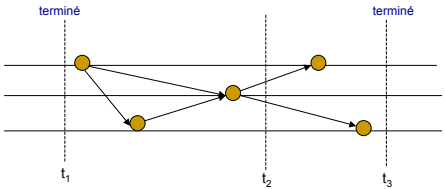
Modèle atomique

- **L'algorithme de détection ne "voit" jamais un processus local dans l'état actif : l'algorithme n'est activé que lorsque le processus est passif**



Modèle atomique

- **Terminaison détectée lorsque tous les canaux son vides.**



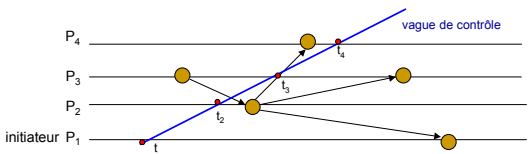
Détection Répartie de la Terminaison

- Modèle atomique :
  - Une **mauvaise solution** avec **deux** compteurs
    - N processus
    - Supposons qu'un processus ***i*** (initiateur) veut savoir si le système se trouve dans un état terminal : tous les canaux vides
      - *i* envoie un message de contrôle à tous les N-1 autres processus à un instant *t*.
    - Chaque processus *j* répond à *i* avec le nombre de messages reçus $r_j(t)$ et nombre de messages envoyés $s_j(t)$;
    - En recevant tous les messages, le site i calcule :
      - $S(t) = \Sigma\, s_j(t_j)$ et $R(t) = \Sigma\, r_j(t_j)$
      - Si S(t) = R(t), le nombre de messages envoyés = nombre de messages reçus alors

        - les canaux sont vides => **détection de la terminaison** **FAUX !!!**
    - **Pourquoi?**

Détection Répartie de la Terminaison

- Inexistence d'un temps global absolu: le moment où les processus *j* ont reçu les messages de contrôle est t_j et non pas *t*, le moment de l'envoi du message de contrôle par *i*.
  - La ligne qui connecte tous les t_j forme une vague de contrôle ("a time cut") .



$s_1(t)=0$; $s_2(t_2)=0$; $s_3(t_3)=1$; $s_4(t_4)=0$;
$r_1(t)=0$; $r_2(t_2)=0$; $r_3(t_3)=0$; $r_4(t_4)=1$;

$S(t) = \Sigma\, s_i(t_i) = 1 = R(t) = \Sigma\, r_i(t_i) = 1$

S(t) = R(t) : canaux vides : Détection de la terminaison => FAUX !!!

Détection Répartie de la Terminaison

- **Solution : L'algorithme des quatre compteurs**
  - Mattern [1987].
  - Compter deux fois :
    - Fin de la première vague de contrôle: l'initiateur accumule les valeurs de $s_i(t_i)$ et $r_i(t_i)$ $\forall$ i : $1 \le i \le N$ dans S* et R*.
    - Fin de la deuxième vague de contrôle: l'initiateur accumule les valeurs de $s_i(t_i)$ et $r_i(t_i)$ $\forall$ i : $1 \le i \le N$ dans S'* et R'* (depuis le début de la première vague).
  - L' exécution est terminé si :

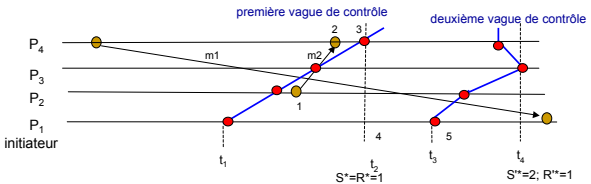
$$\mathbf{S^* = R^* = S'^* = R'^*}$$

    - L'exécution est terminée à la fin de la première vague.

Détection Répartie de la Terminaison

L'algorithme des Quatre Compteurs

Application n'a pas terminé : S*=R*=R'*=1 mais S'*=2

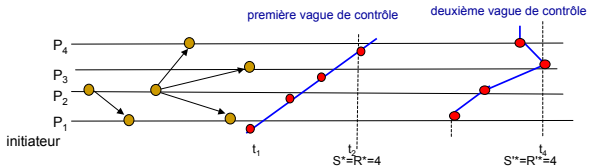


● : Site P_i reçoit le msg de contrôle de P_1 et renvoi les information sur $s(t_2)_i$ et $r(t_2)_i$
Deuxième vague commence après la reception de tous les messages de contrôle: après t_2

Détection Répartie de la Terminaison

L'algorithme des Quatre Compteurs

Application a terminé: S*=R*=S'*=R'*=4



R* = S'* => l'exécution s'est terminée à la fin de la première vague: t_2
Terminaison détectée à la fin de la deuxième vague : t4

Détection Répartie de la Terminaison

- **L'algorithme des quatre Compteurs (cont.)**
  - R*= S'*, alors l'exécution répartie s'est terminée à la fin de la première vague.
    - Soient t_2 la date où la première vague s'est terminée et $t_3 \geq t_2$ la date du début de la deuxième vague.

$$R^* = S'^* \Rightarrow R(t_2) = S(t_2)$$

Détection Répartie de la Terminaison

- **L'algorithme des quatre Compteurs**

(1) Les compteurs locaux sont monotones, $t \leq t'$ implique $s_i(t) \leq s_i(t')$ et $r_i(t) \leq r_i(t')$.
- *Preuve* : suit de la définition.

(2) Le nombre de messages envoyés et reçus est monotones, $t \leq t'$ implique $S(t) \leq S(t')$ et $R(t) \leq R(t')$.
- *Preuve* : suit de la définition et (1).

(3) $R^* \leq R(t_2)$.
- *Preuve* : suit de (1) et le fait que toutes les valeurs de r_i sont collectées avant ($\leq$) t_2.

(4) $S'^* \geq S(t_3)$.
- *Preuve* : suit de (1) et le fait que toutes les valeurs de s_i sont collectées après ($\geq$) t_3.

(5) $\forall$ t, $R(t) \leq S(t)$.
- *Preuve*: la différence non négative $D(t) = S(t) - R(t)$ correspond au nombre de messages en transit. $D(t) \geq 0$.

Détection Répartie de la Terminaison

- **L'algorithme des Quatre Compteurs**

$$R^* = S'^* \Rightarrow R(t_2) \geq S(t_3) \quad (3,4)$$
$$\Rightarrow R(t_2) \geq S(t_2) \quad (2)$$
$$\Rightarrow R(t_2) = S(t_2) \quad (5)$$

- Cela dit, l'exécution s'est terminée à l'instant t_2

Détection Répartie de la Terminaison

- **Bibliographie**
  - J. Misra, Detecting termination of distributed computations using markers. PODC, pages 290-294.
  - E.W.Dijkstra, Derivation of a termination detection algorithm for distributed computations. *Information Processing Letters* 16, pages 217-219, 1983
  - F. Mattern, Algorithms for distributed termination detection. *Distributed Computing,* Vol 2, pages 161-175, Springer–Verlag, 1987.
  - S. P. Rana, A distributed solution of distributed termination problem. *Information Processing Letters* 17, pages 43-46, 1983.
  - J. Matocha and T. Camp, A taxonomy of distributed termination detection algorithms. *The Journal of Systems and Softwares 43*, pages 207-221, 1998.

Infrastructures (overlays)
Auto*
Construction et Applications

Maria.Gradinariu@lip6.fr

Infrastructures - exemples

- Internet
- Réseaux P2P - infrastructures logiques
  - non-structurées
  - structurées (Distributed Hash Tables)
- Réseaux Mobiles & Réseaux de capteurs
  - Clustering – cellules, pico-cellules
  - Couvertures connexes
  - backbones

Infrastructures - applications

- Router et rechercher des données et services
  - Gnutella, Kazaa, BitTorrent
- Communiquer et collaborer
  - Chat/Irc, NewsGroups
  - Instant Messaging (Aol, Icq, Yahoo, Msn)

Infrastructures - applications

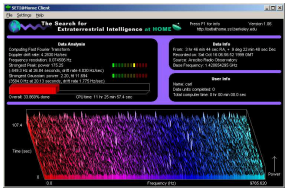
- Partager/répartir des données, bande passante, puissance de calcul ou stockage
  - Napster, Publius, Freenet, MojoNation, FreeHaven, Groove, e-donkey, Chord, Can, Pastry, Tapestry
  - Voice/IP (Skype)
  - Seti@home (astronomie)
  - genome@home (ADN)
  - folding@home (repliement des protéines)

SETI@home
The Search for Extraterrestrial Intelligence

- Expérience en radioastronomie exploitant la puissance inutilisée de millions d'ordinateurs connectés via Internet dans un projet de Recherche d'une Intelligence Extra-terrestre (Search for Extra-Terrestrial Intelligence, alias SETI).
- Les ordinateurs chargent et analysent les données collectées du plus grand radiotélescope au monde à Arecibo.
- 2,4 millions de participants de 226 pays et territoires

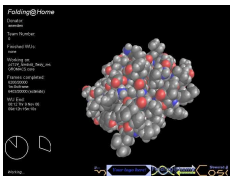
SETI@home
The Search for Extraterrestrial Intelligence

- 3.8M utilisateurs dans 226 pays
- 1200 années CPU / jour
- 38 TeraFlops soutenu (Le Earth Simulator Japonais obtient 40 TF)
- 1.7 Zettaflop (10^{21}) pour les 3 dernières années
- Très hétérogène : >77 types de processeurs differents



Folding@home distributed computing

- Comprendre le repliement et l'agrégation des protéines et les maladies qui sont liées
- Etude de maladies comme celle d'Alzheimer, la fibrose cystique, l'EBS (la Vache Folle), une forme héréditaire de l'emphysème et de nombreux cancers résultent d'un repliement anormal des protéines.
- Depuis le 1er octobre 2000, plus de 500,000 CPUs ont participé a ce programme



Réseaux P2P

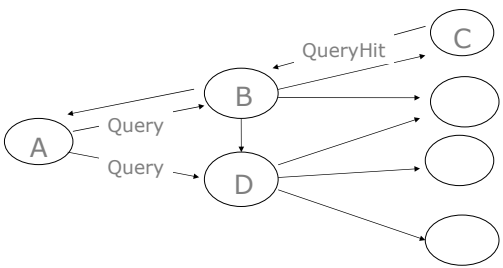
Infrastructures P2P

- non –structurées (Gnutella, Kazaa)
  - la topologie du système est déterminée par les utilisateurs
  - le placement des données dans le système ne tient pas compte de la topologie du système
- structurées (Chord, CAN, Pastry,Tapestry)
  - le placement des données dans le système est fait à des locations précises (utilisation des fonctions de « hash »)
  - la topologie du réseaux a des formes particulières (ex. anneau, arbre, grille)

Réseaux P2P non-structurées Gnutella

- Protocole de recherche de données et services
  - chaque nœud est à la fois client et serveur
- Messages Gnutella (TimeToLive)
  - découverte de nœuds PING/PONG
  - découverte de données (fichiers) et services
    - Query
    - QueryHit

Réseaux P2P non-structurées Gnutella



Réseaux P2P semi-structurées FreeNet

- Stockage persistant de données et services
  - Données identifiées par une clé binaire (fonction hash)
  - les données qui traversent un nœud sont copiées dans le cache du nœud
    - L'utilisation de la politique LRU pour la gestion du cache
    - L'information stockée par donnée (fichier)
      - code hash
      - Le dernier temps d'accès/modification

Réseaux P2P semi-structurées
FreeNet

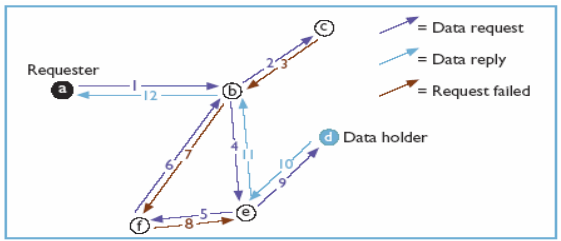


Figure 1.Typical request sequence.The request moves through the network from node to node, backing out of a dead-end (step 3) and a loop (step 7) before locating the desired file.

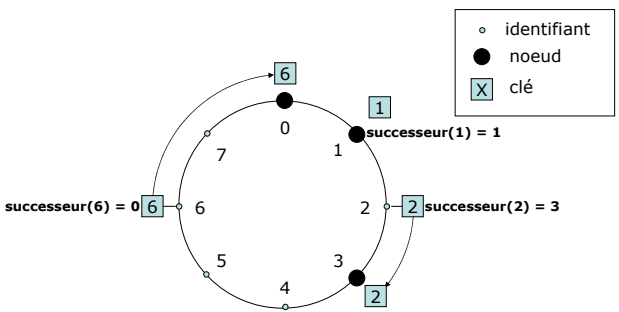
Gnutella vs. FreeNet

- Routage basé sur la diffusion (flooding) vs. Routage dynamique basé sur la similarité des clés
- Aucune mémoire du trafique passé vs. Tables de routage
- Read-only vs. Read/Write
- Système non sécurisé vs. Système sécurisé

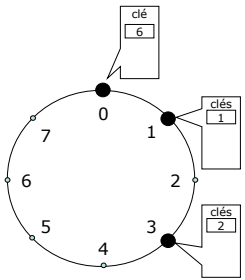
Réseaux P2P structurées
Chord

- une infrastructure de stockage et routage
- à chaque utilisateur et à chaque fichier dans le système on associe un identifiant sur m bits (2^m identifiants)
  - Id utilisateur = hash(IP)
  - Id fichier ou clé fichier = hash(contenu)
  - L'espace des IDs organisé en anneau
  - Un fichier de clé k est stocké sur le premier noeud dans le système ayant l'identifiant y (y>k mod 2^m)

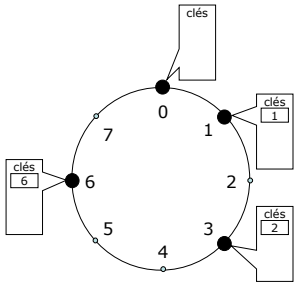
Réseaux P2P structurées
Chord (l'association clés - noeuds)



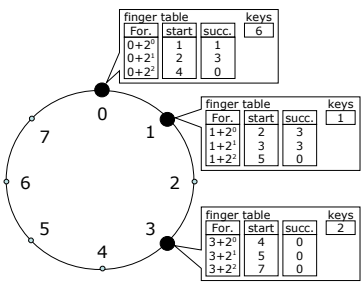
Réseaux P2P structurées
Chord (l'association clé - noeuds)



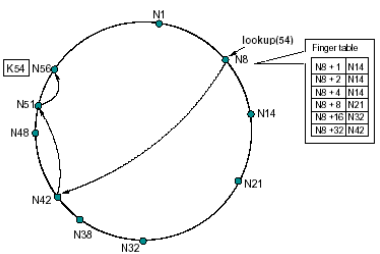
Réseaux P2P structurées
Chord (l'entrée du nœud 6)



Réseaux P2P structurées
Chord (les raccourcis)



Réseaux P2P structurées
Chord (la recherche de la clé 54)



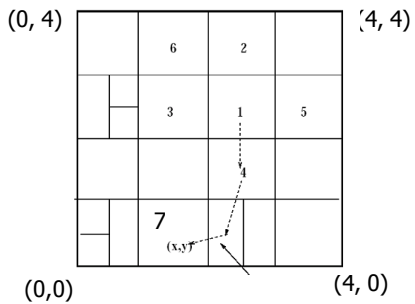
Réseaux P2P structurées
Chord

- la mémoire utilisée par noeud O(log(N))
- le temps de recherche d'une clé O(log(N))
- le système est auto-reconfigurable et tolérant aux défaillances

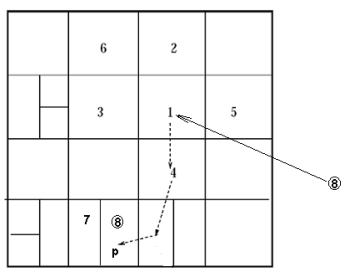
Réseaux P2P structurées
CAN

- Idée de conception :
  - espace cartésien virtuel 2-dimesionnel
  - chaque nœud du système est le propriétaire d'une zone dans l'espace virtuel
  - les données sont stockées sous la forme (clé, val)
    - hash(clé) --> un point (x,y) dans l'espace virtuel
    - (clé, val) est stocké par le nœud propriétaire de la zone dont fait partie (x,y)

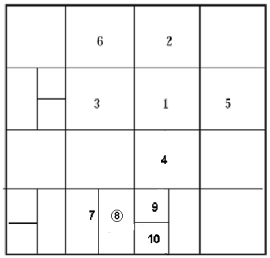
Réseaux P2P structurées
CAN (le routage)



Réseaux P2P structurées
CAN (l'insertion du nœud 8)



Réseaux P2P structurées
CAN (le départ du nœud 9)



Infrastructures P2P
Réplication de données

- FreeNet – les données sont copiées par les nœuds qui participent à leur routage
- MojoNation – les copies des données très demandées sont disséminées dans le réseau par un serveur
- CAN (multi-dimensionnel) – une donnée peut avoir une clé par dimension

Infrastructures P2P
Sécurité


"On the Internet, nobody knows you're a dog."

Infrastructures P2P
Sécurité

- éviter les connexions directes entre le demandeur d'information et le propriétaire (FreeNet)
- utiliser des TTL choisis aléatoirement
- dissocier le propriétaire d'un document de l'emplacement où le document est stocké (CAN, Chord)

Infrastructures P2P
Sécurité

- la vérification des données
  - clés cryptographiques (CFS, Past)
- la dissémination des données
  - les fichiers à stocker sont décomposés en n blocs de telle manière à ce que m blocs (m<n) sont suffisants pour reconstituer le fichier (Publius, Mnemosyne, FreeHaven)

Infrastructures P2P
Sécurité

- « free-riding » et collusion
  - un ou plusieurs utilisateurs profitent du système sans partager leur ressources
  - Problème : écroulement du système
  - Solutions :
    - Utilisation des technique d'incitation à la participation
      - Découverte des ressources du système proportionnelle à la participation
      - Payement virtuel ou micro-payement (MojoNation)
    - Surveiller les pairs

Infrastructures P2P
Sécurité

- « sybil attack » [Douceur 2002]
  - Un utilisateur peut entrer dans le réseau en utilisant plusieurs identités
  - Problèmes dans les systèmes qui utilisent la réplication ou la fragmentation de données
  - Solution (utopique) : l'identification unique des ressources d'un nœud