Systèmes pair-à-pair

Frédéric Vivien

e-mail: Frederic.Vivien@ens-lyon.fr

9 décembre 2005

Plan du cours

Plan du cours

Définition

Définition

Un ensemble de participants indépendants qui mettent en commun des ressources afin d'aboutir à un résultat/exécuter une tâche. Ces participants exécutent un programme au but identique.

Il n'y a pas une entité indépendante des autres qui gère le service.

Avantages

- L'augmentation du nombre d'utilisateurs entraîne l'augmentation des ressources disponibles.
- Difficile d'attaquer le système.

Contraintes

- Machines fortement hétérogènes.
- Connexions fortement hétérogènes.
- Connexions intermittentes.
- Connaissances uniquement locales.

Caractéristiques des systèmes pair-à-pair

- Extensibilité de plusieurs milliers à plusieurs millions de machines.
- Hétérogénéité (puissance, mémoire, connexion, OS, etc.)
- Dynamicité : utilisation intermittente.
- Tolérance aux pannes
- Tolérance aux attaques
- Sécurité : il ne faut pas qu'une entité puisse prendre le contrôle du système.
- Adaptation à l'utilisation : gestion des *hotspots*.
- Configurabilité

Propriétés secondaires

- Anonymat : *routage oignon* ou prise de requête, cryptographie.
- Pérennité des données : répliquer les données et rendre leur localisation difficile.
- Stockage : les systèmes pair-à-pair peuvent être des systèmes de stockage. (Forte pérennité $\neq$ optimiser l'accessibilité des données « favorites ».)

Plan du cours

Systèmes clients-serveurs.

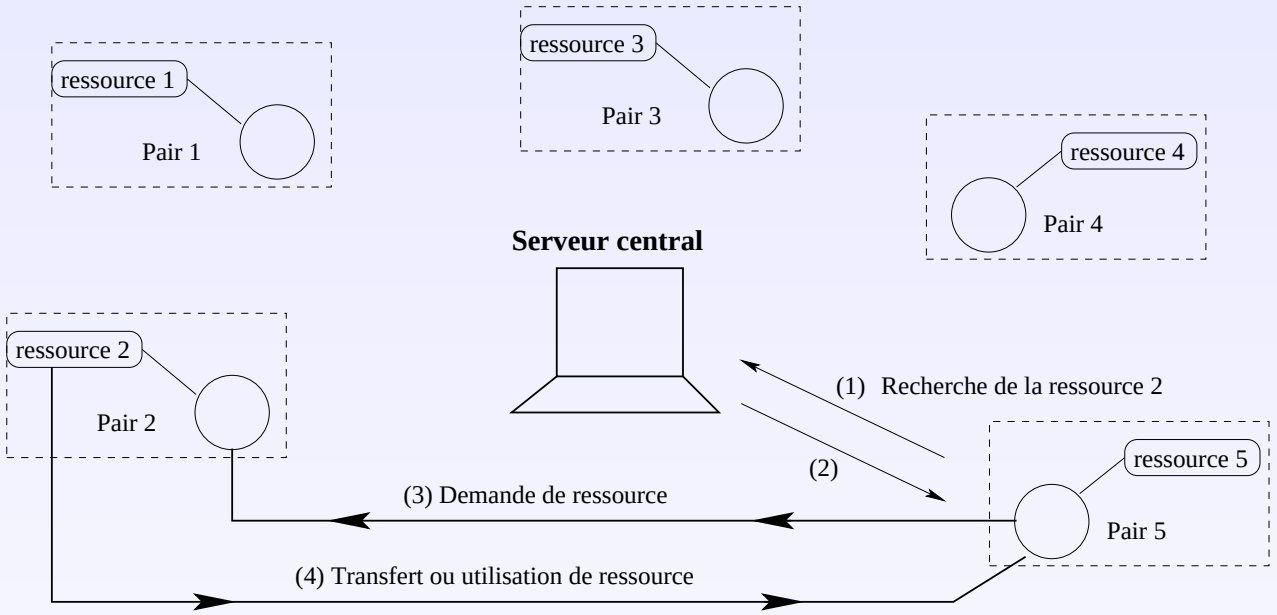
(Pas un système pair-à-pair mais sert de point de comparaison.)

Le client est un consommateur pur : le client communique uniquement avec le serveur pour que le serveur lui communique une information ou exécute un service.

Exemple : NFS (Network File System).

- Norme de partage centralisé de documents.
- Les ressources de stockage sont celles du serveur.
- Pas de communications entre les clients.

Les systèmes d'indexation centralisés (1)



- Les participants doivent se connecter à un serveur central avant de se connecter au réseau.
- Le serveur est une entité de localisation pure :
  - pas de ressources ;
  - permet de référencer les participants et les ressources ;
  - les communications ont lieu uniquement entre participants.

Les systèmes d'indexation centralisés (2)

Les systèmes d'indexation centralisés (2)

Avantages

- Localisation rapide et performante.
- Mises à jour faciles et régulières.

Les systèmes d'indexation centralisés (2)

Avantages

- Localisation rapide et performante.
- Mises à jour faciles et régulières.

Inconvénients

- Tout le système repose sur le seul serveur (fragilité, goulot d'étranglement).
  En pratique : une liste de serveurs.

Les systèmes d'indexation centralisés (2)

Avantages

- Localisation rapide et performante.
- Mises à jour faciles et régulières.

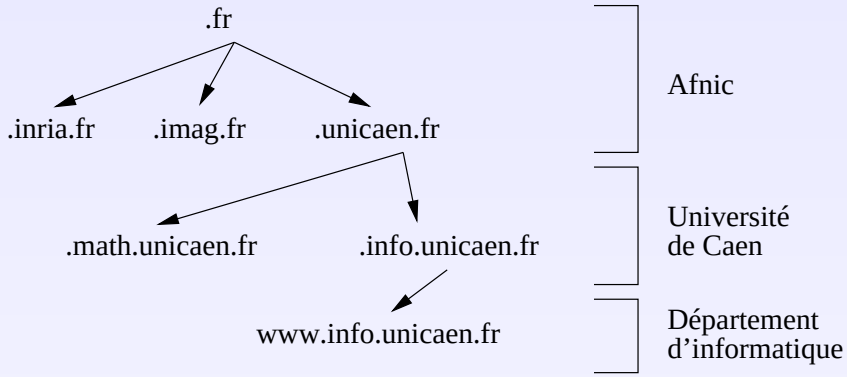
Inconvénients

- Tout le système repose sur le seul serveur (fragilité, goulot d'étranglement).
  En pratique : une liste de serveurs.

Exemples

- Seti@Home, Folding@Home
- Napster
  Autre inconvénient : pas de pérennité des données.

Les systèmes hiérarchiques



Exemple : le système des noms de domaine

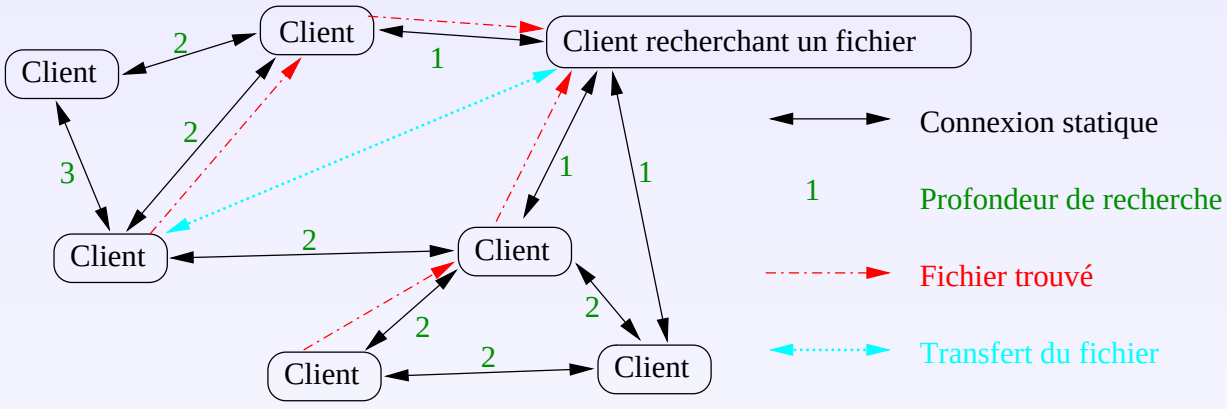
- La responsabilité des sous-ensembles de noms est partagée entre plusieurs organisations : arbre de responsabilité.
- Quand un client (extérieur) veut connaître une adresse :
  1. Il envoie une requête à un serveur racine ;
  2. Le serveur racine le renvoie sur un serveur DNS plus bas dans la hiérarchie et gérant le domaine contenant le nom recherché.
- Système de cache pour limiter le flot de messages.
- Interrogation du cache local en premier.
- Système distribué à contrôle centralisé.

Les systèmes par inondation : Usenet

Usenet (forums de discussion)

- Système distribué sur un grand nombre de serveurs.
- Lorsqu'un serveur reçoit un message, il le transmet à tous ses voisins qui font de même.
- La structure de voisinage est construite par les administrateurs des serveurs Usenet.
- Chaque serveur possède à tout moment presque tous les messages récents.

Les systèmes par inondation : Gnutella (1)



Fonctionnement d'un système Gnutella. La requête passe par tous les participants du système, et tout ceux qui possèdent un fichier correspondant répondent. Ensuite le nœud initial choisi l'un d'eux et effectue le transfert.

Les systèmes par inondation : Gnutella (2)

- Pour se connecter il faut connaître un participant.
- Pour chercher une donnée, on envoie une requête à ses voisins, qui la retransmettent en diminuant sa durée de vie.
- Transfert direct entre détenteur et demandeur.

Les systèmes par inondation : Gnutella (2)

- Pour se connecter il faut connaître un participant.
- Pour chercher une donnée, on envoie une requête à ses voisins, qui la retransmettent en diminuant sa durée de vie.
- Transfert direct entre détenteur et demandeur.

Inconvénients

- Une requête pour un document existant peut ne pas aboutir (document trop loin, pic de popularité).
  Ajout de caches (de résultats de requêtes).
- Très coûteux en terme de communications.
- Problème de confidentialité : très facile de savoir qui (adresse IP) possède quoi.
- 70% des utilisateurs ne partagent rien.

Les systèmes structurés

- Fournissent un routage efficace d'un point à un autre du système. (Pas besoin d'inonder le système.)
- Basés sur une notion de connaissance locale.
  Un nœud n'a pas de connaissance globale : mais il peut se rapprocher de la donnée recherchée.

Exemples : Pastry/Tapestry, Silverback, Can, Chord, etc.

Plan du cours

Définition

- Le protocole Chord ne sert qu'à localiser des clés.

Définition

- Le protocole Chord ne sert qu'à localiser des clés.
- La fonction de hachage SHA-1 sert à attacher un *identificateur* sur m bits à chaque nœud et clé.
  m est choisi suffisamment grand pour que la probabilité d'une collision soit suffisamment petite.
  Identificateur d'un nœud : image de son adresse IP par SHA-1.
  Identificateur d'une clé : son image par SHA-1.

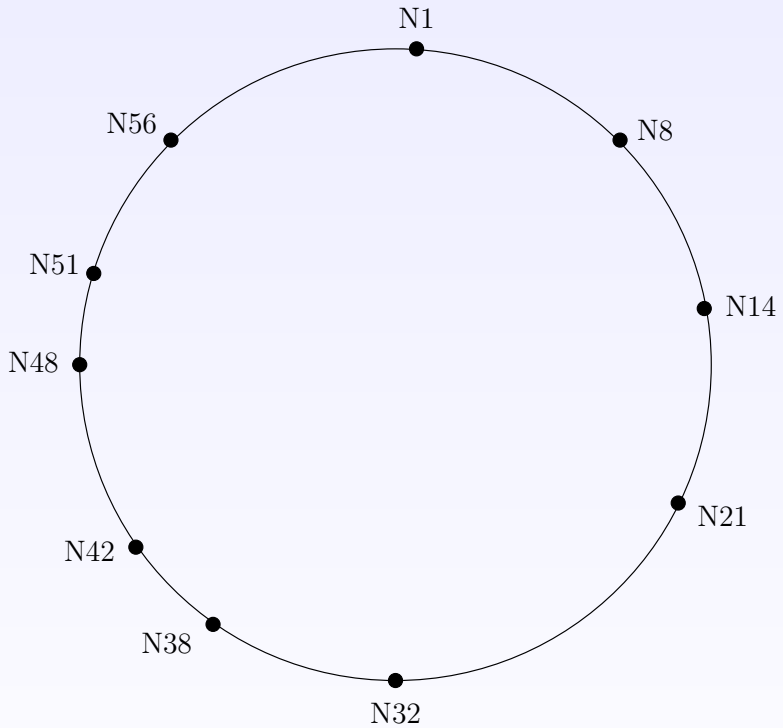
Définition

- Le protocole Chord ne sert qu'à localiser des clés.
- La fonction de hachage SHA-1 sert à attacher un *identificateur* sur m bits à chaque nœud et clé.
  m est choisi suffisamment grand pour que la probabilité d'une collision soit suffisamment petite.
  Identificateur d'un nœud : image de son adresse IP par SHA-1.
  Identificateur d'une clé : son image par SHA-1.
- Les identificateurs sont ordonnées sur le cercle des identificateurs modulo 2^m.

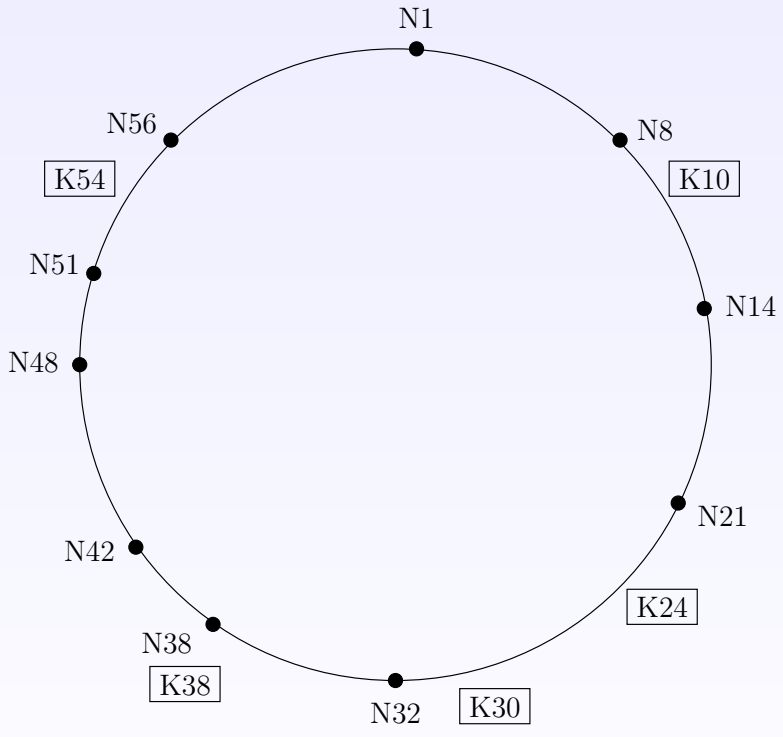
Définition

- Le protocole Chord ne sert qu'à localiser des clés.
- La fonction de hachage SHA-1 sert à attacher un *identificateur* sur m bits à chaque nœud et clé.
  m est choisi suffisamment grand pour que la probabilité d'une collision soit suffisamment petite.
  Identificateur d'un nœud : image de son adresse IP par SHA-1.
  Identificateur d'une clé : son image par SHA-1.
- Les identificateurs sont ordonnées sur le cercle des identificateurs modulo 2^m.
- La clé k est localisée sur le premier nœud dont l'identificateur est supérieur ou égal à (l'identificateur de) k.
  La clé k est localisée sur le nœud $successeur(k)$.

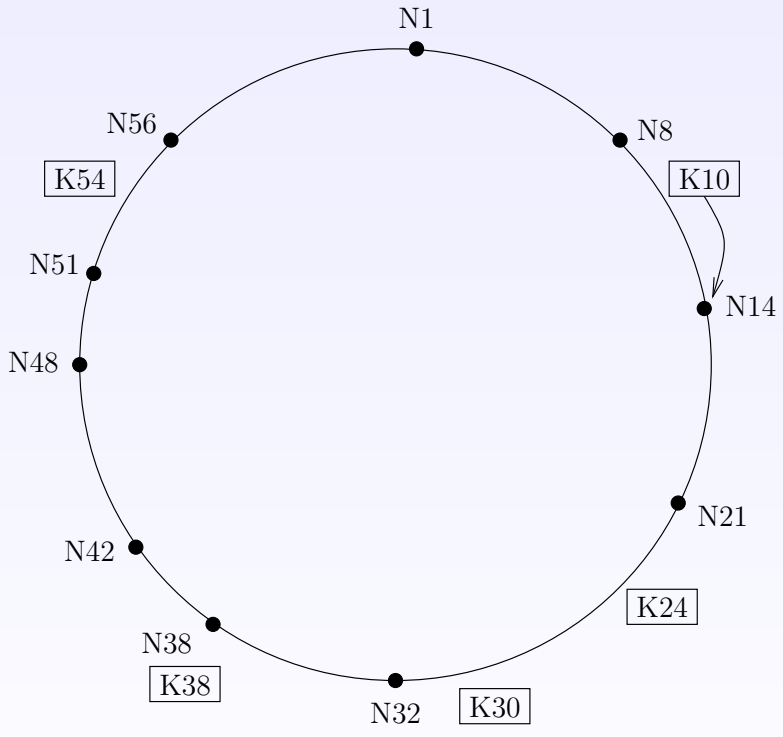
Exemple d'anneau de Chord



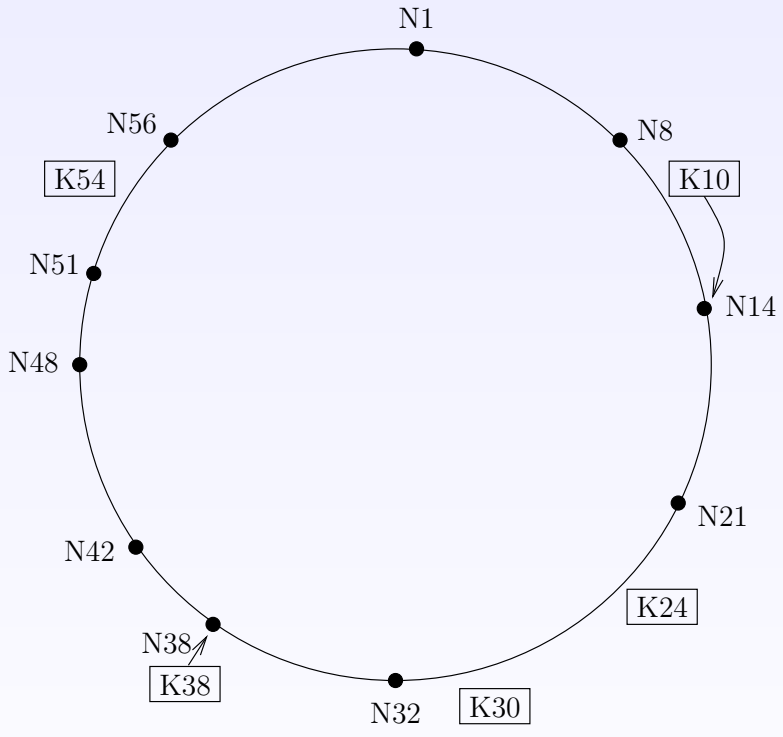
Exemple d'anneau de Chord



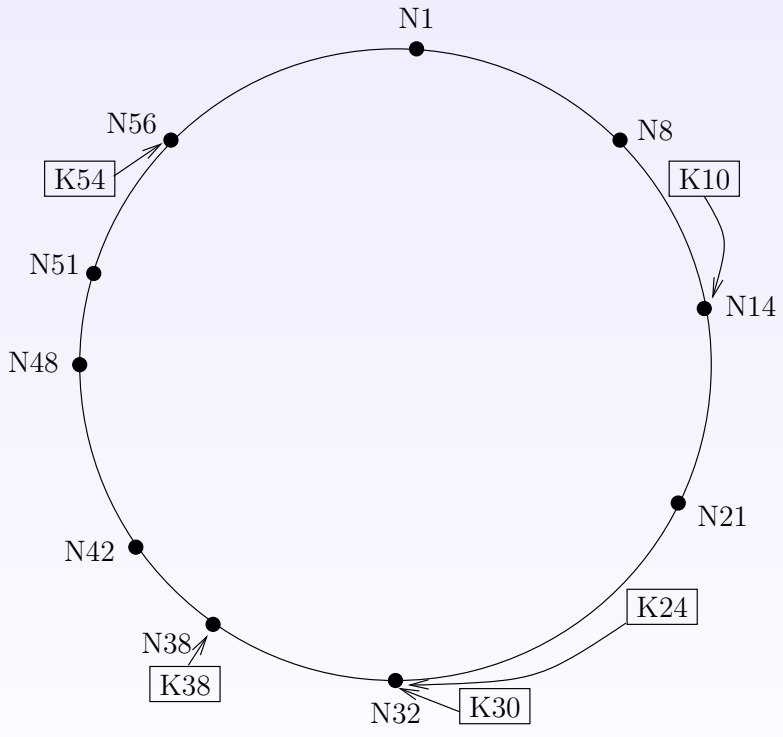
Exemple d'anneau de Chord



Exemple d'anneau de Chord



Exemple d'anneau de Chord



Répartition des clefs

- Quand un nœud n rejoint le réseau, certaines des clefs précédemment localisées sur $successeur(n)$ sont relocalisées sur n.
- Quand un nœud n quitte le réseau, toutes les clefs précédemment localisées sur n sont relocalisées sur $successeur(n)$.

Répartition des clefs

- Quand un nœud n rejoint le réseau, certaines des clefs précédemment localisées sur $successeur(n)$ sont relocalisées sur n.
- Quand un nœud n quitte le réseau, toutes les clefs précédemment localisées sur n sont relocalisées sur $successeur(n)$.

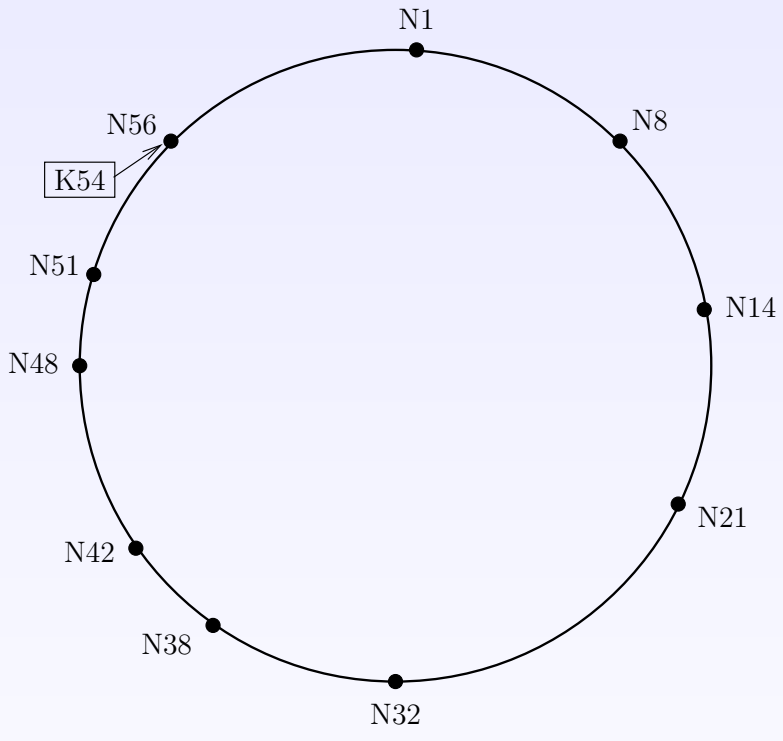
Théorème

Pour un système de N nœuds et K clés, avec une très grande probabilité :

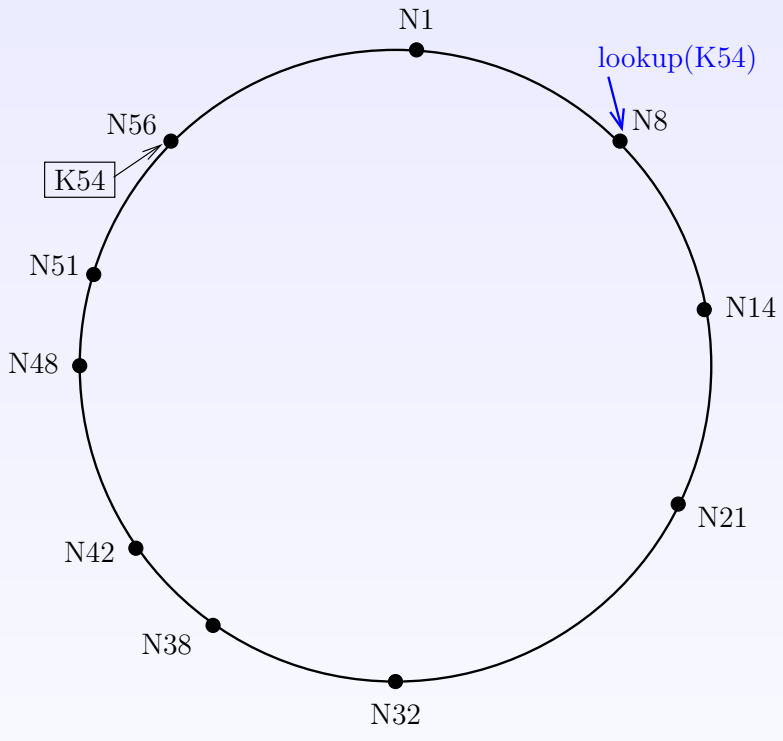
1. *Chaque nœud est responsable d'au plus $(1+\epsilon)\frac{K}{N}$ clés.*
2. *Quand un $N+1^e$ nœud rejoint ou quitte le réseau, $O(\frac{K}{N})$ nœuds changent de mains.*

Ici, $\epsilon = O(\log N)$.
ϵ peut être rendu arbitrairement petit en plaçant $\Omega(\log N)$ nœuds virtuels, avec leurs propres identificateurs, sur chaque nœud.

Recherche d'une clé : méthode simple

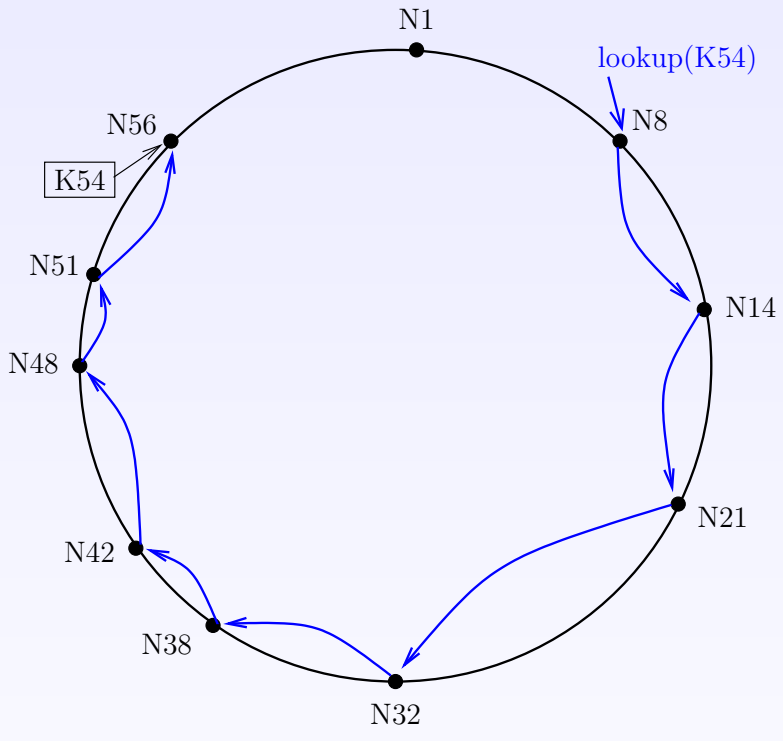


Recherche d'une clé : méthode simple



Le nœud 8 recherche la clef 54.

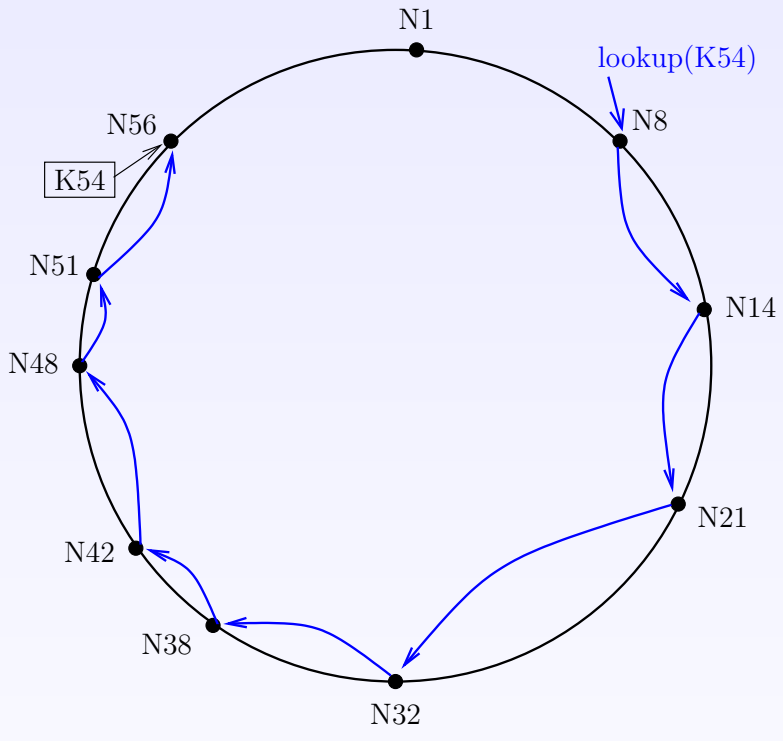
Recherche d'une clé : méthode simple



Le nœud 8 recherche la clef 54.

La recherche visite tous les nœuds entre les nœuds 8 et 56.

Recherche d'une clé : méthode simple



Le nœud 8 recherche la clef 54.

La recherche visite tous les nœuds entre les nœuds 8 et 56.

Complexité : $O(N)$.

Recherche d'une clé : définition des *fingers* (1)

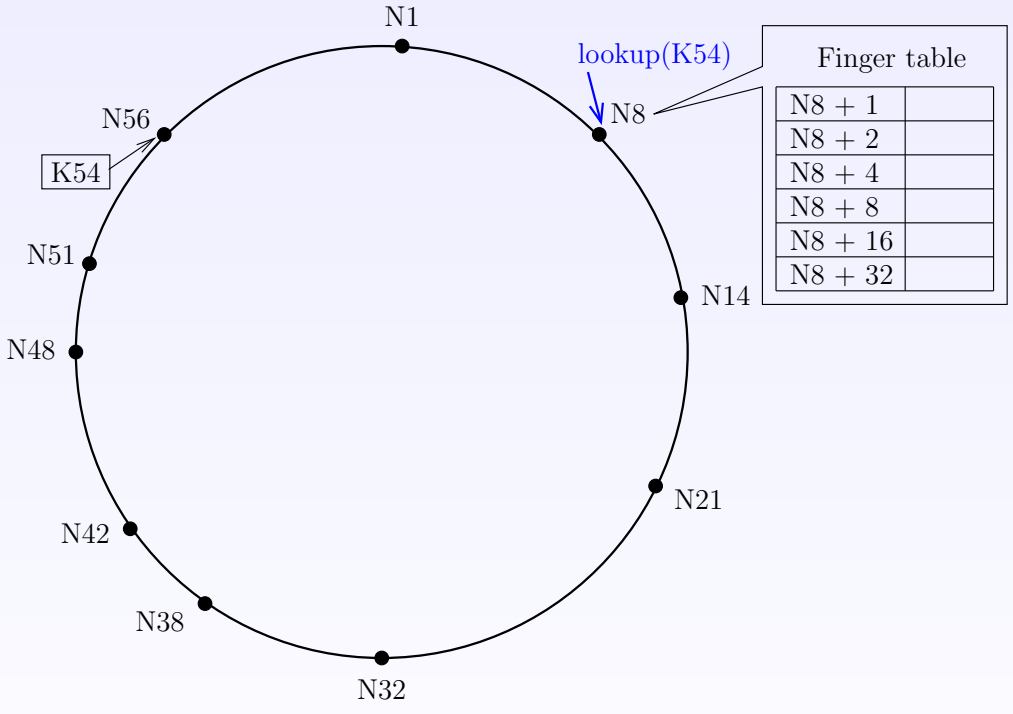
Plus d'informations de routages pour accélérer la recherche.

- Chaque nœud a une table de routage à m entrées
  C'est la table des *fingers* (pointeurs).
- La i^{e} entrée : identité du nœud s, premier nœud qui est au moins 2^{i-1} après le nœud courant (n).

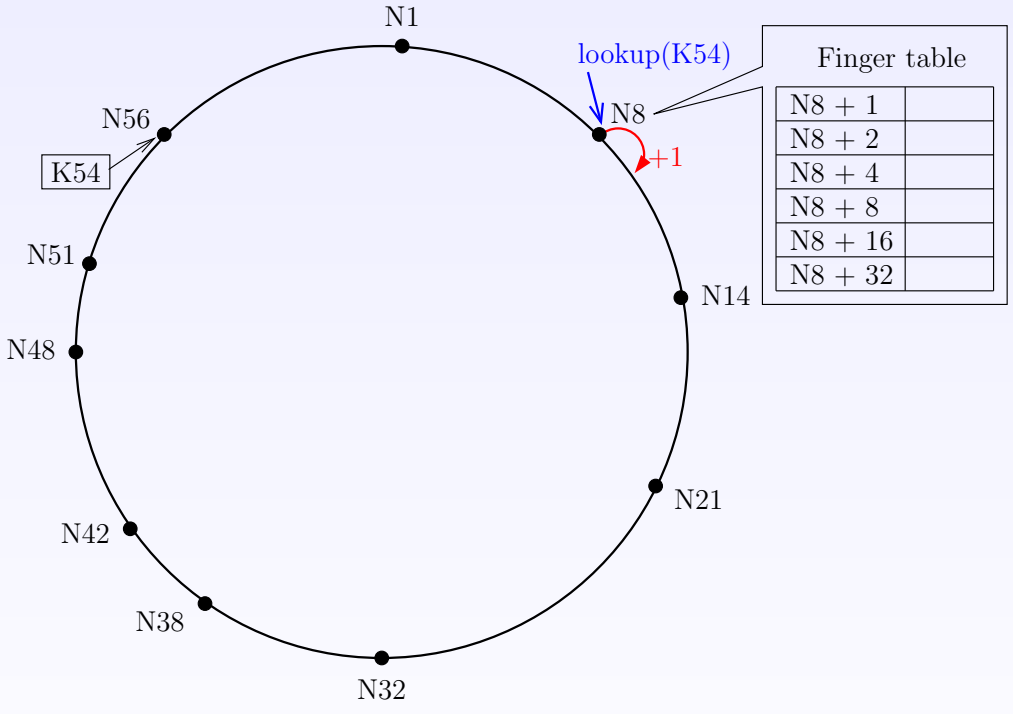
$$n.finger[i] = successeur(n + 2^{i-1}).$$

- En pratique : la i^{e} entrée contient l'identité et l'adresse IP du nœud en question.

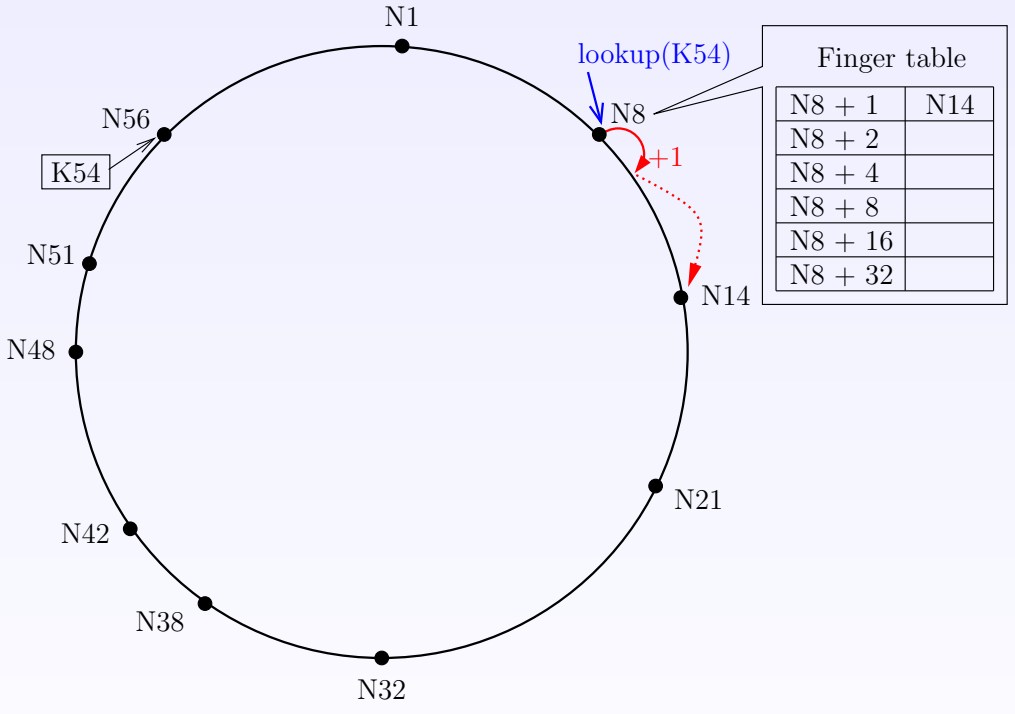
Recherche d'une clé : définition des *fingers* (2)



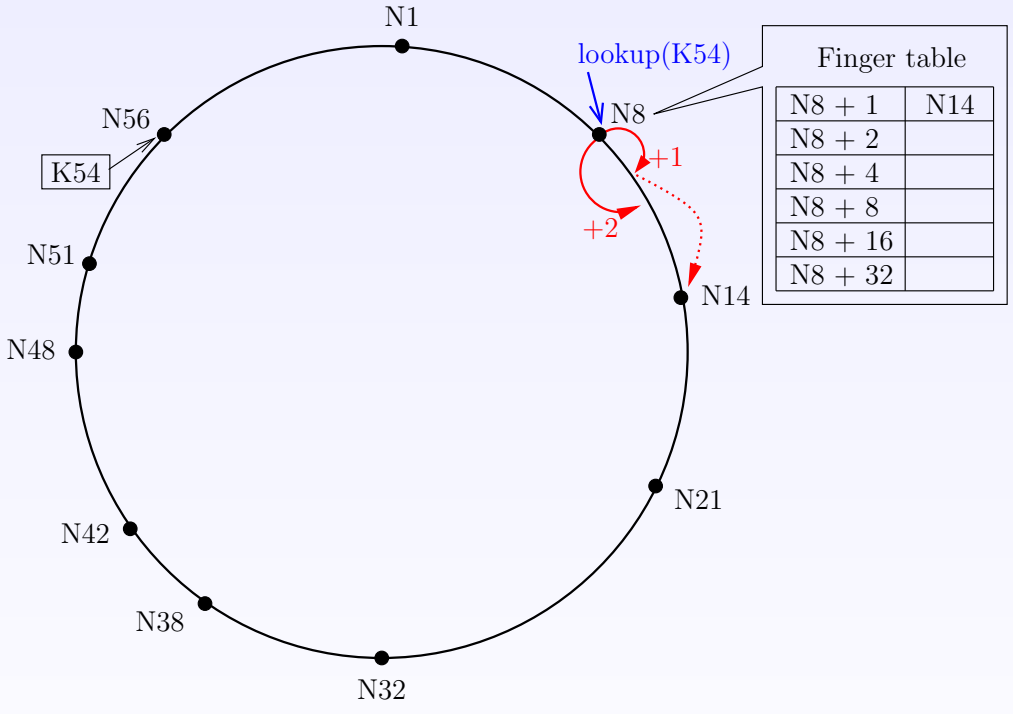
Recherche d'une clé : définition des *fingers* (2)



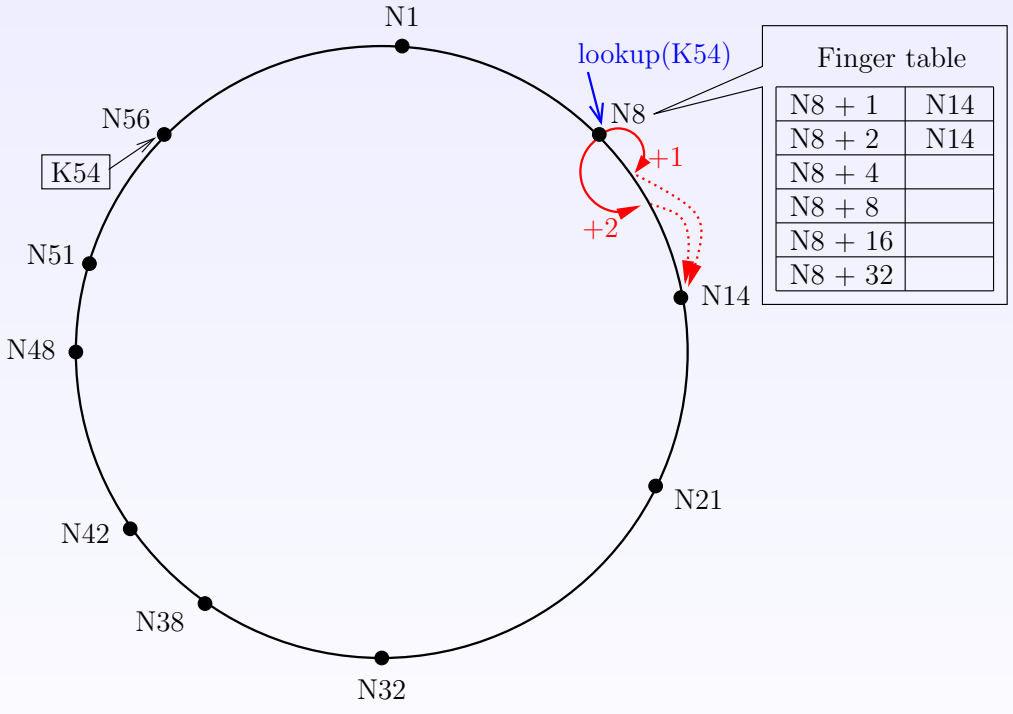
Recherche d'une clé : définition des *fingers* (2)



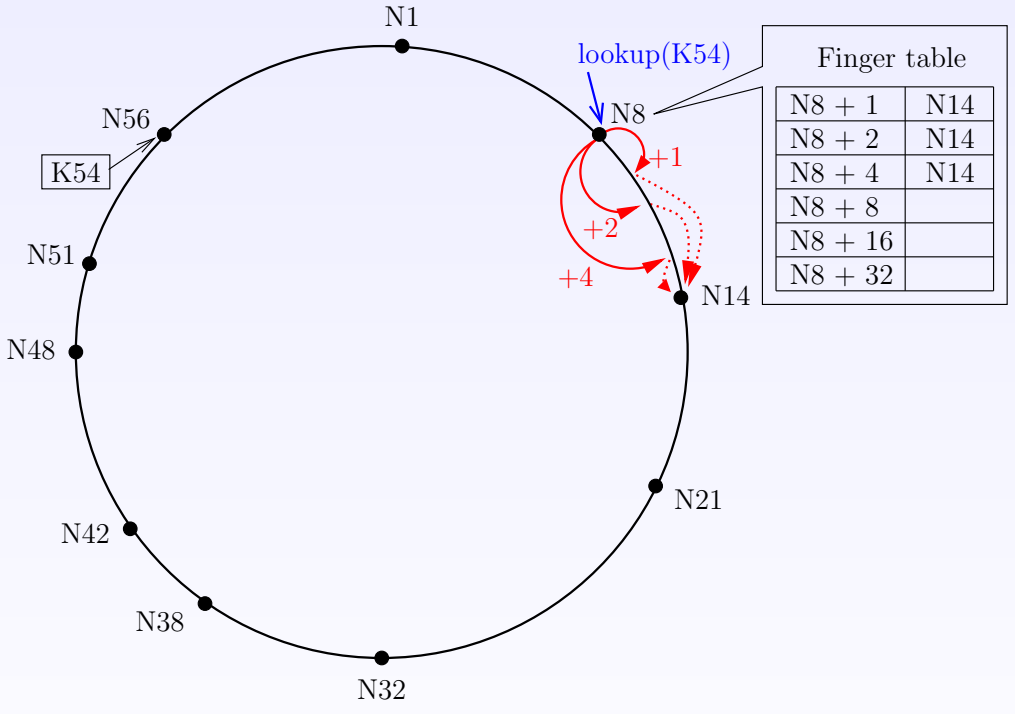
Recherche d'une clé : définition des *fingers* (2)



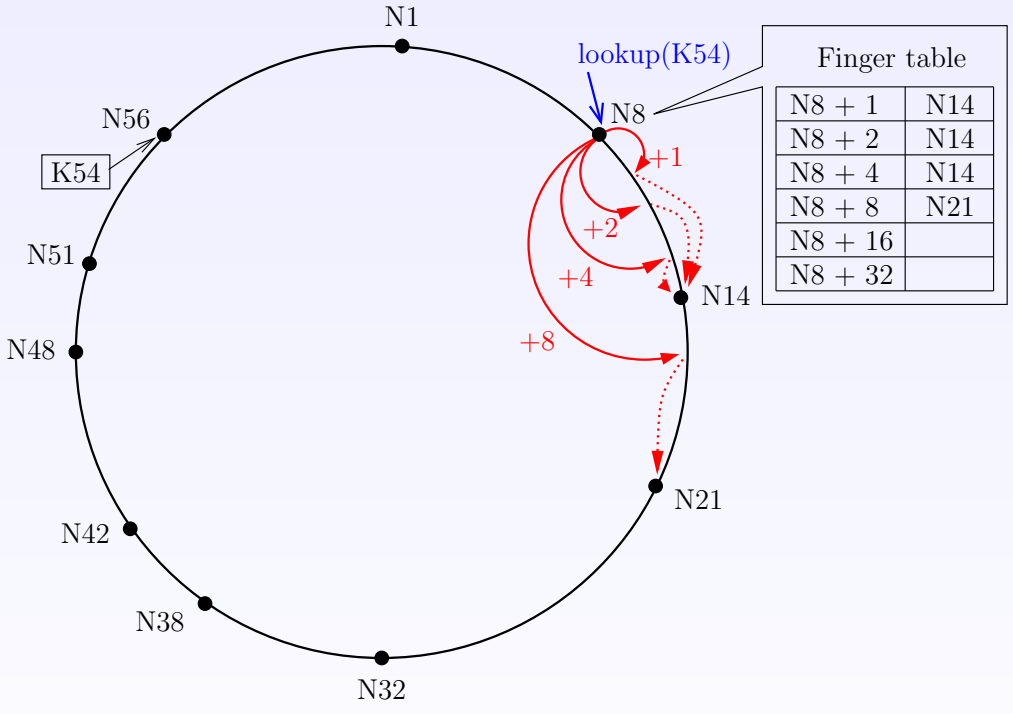
Recherche d'une clé : définition des *fingers* (2)



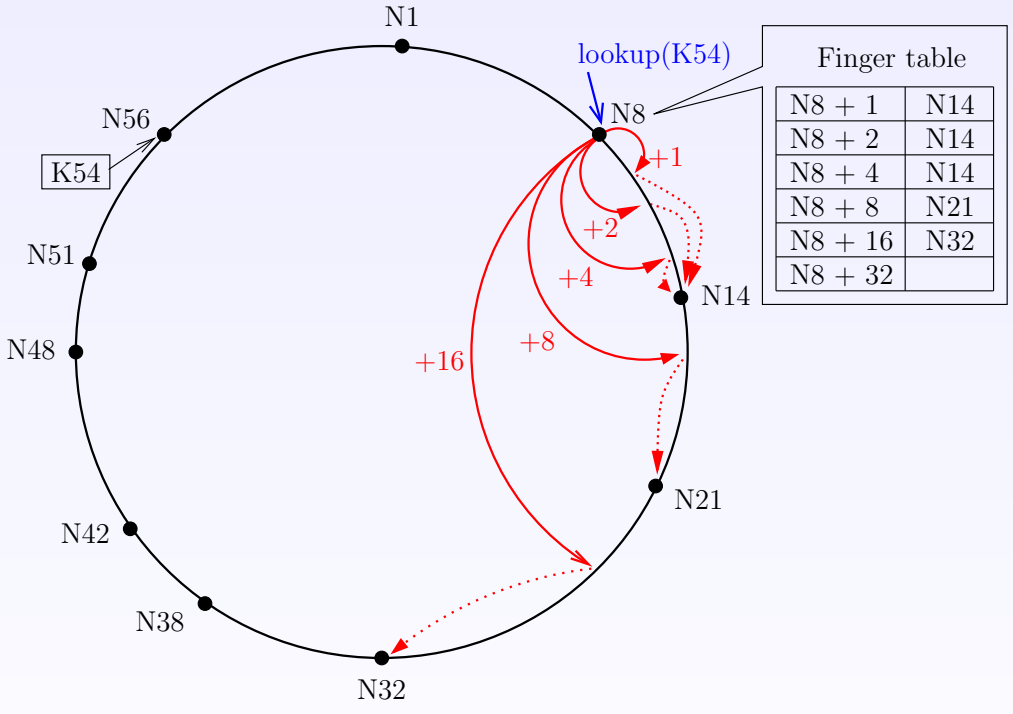
Recherche d'une clé : définition des *fingers* (2)



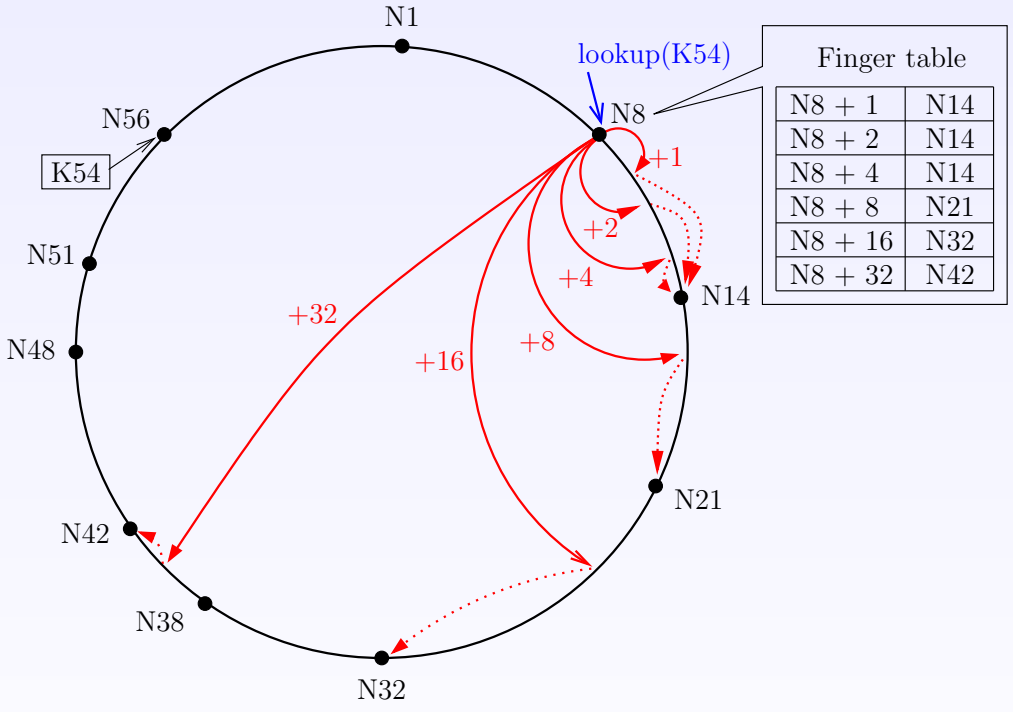
Recherche d'une clé : définition des *fingers* (2)



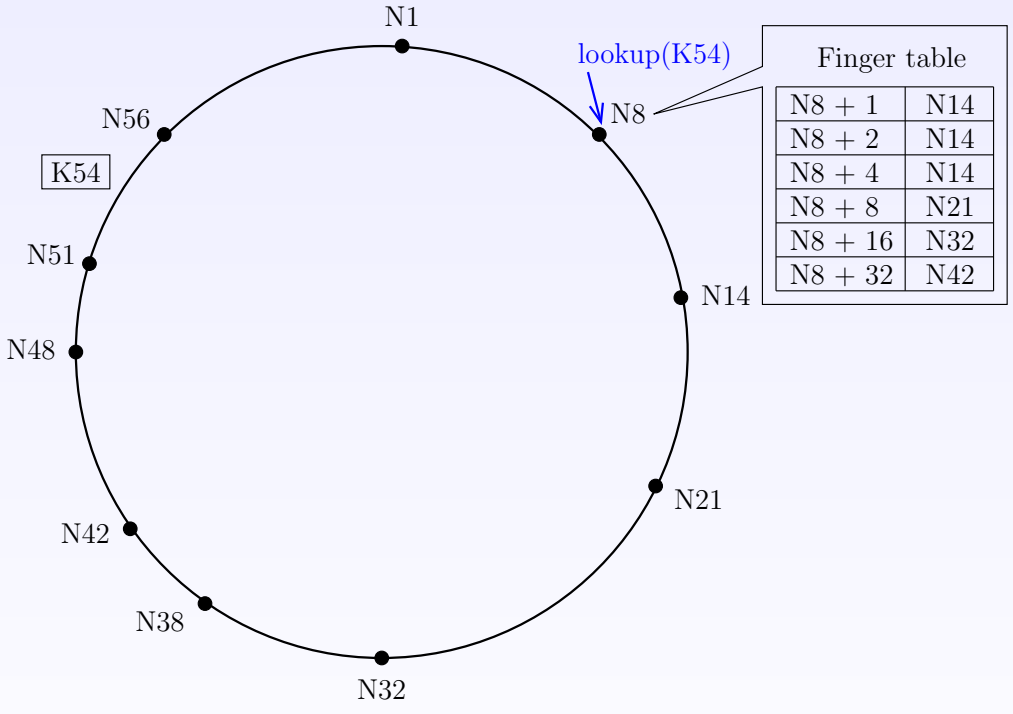
Recherche d'une clé : définition des *fingers* (2)



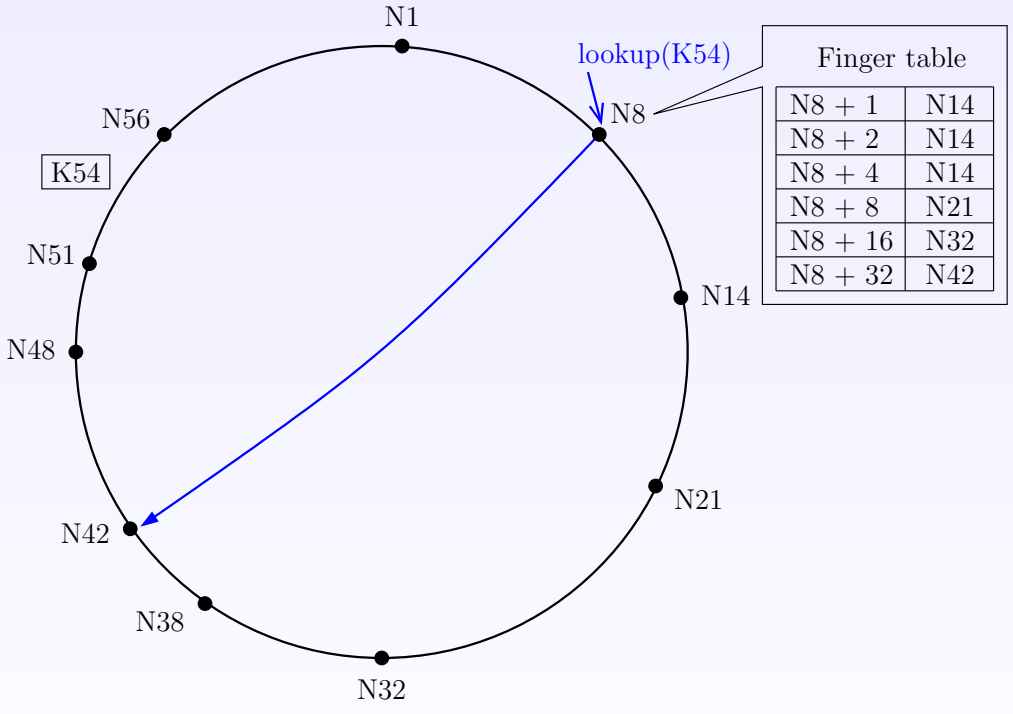
Recherche d'une clé : définition des *fingers* (2)



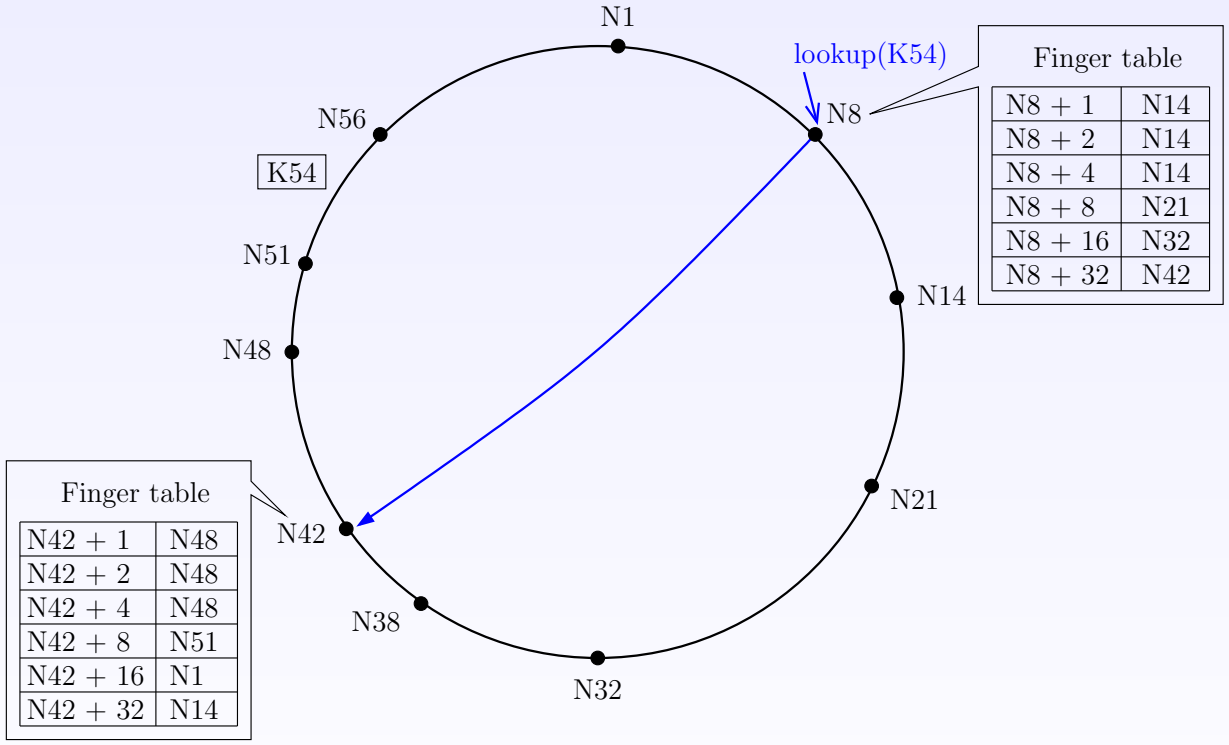
Recherche d'une clé au moyen des *fingers* : exemple



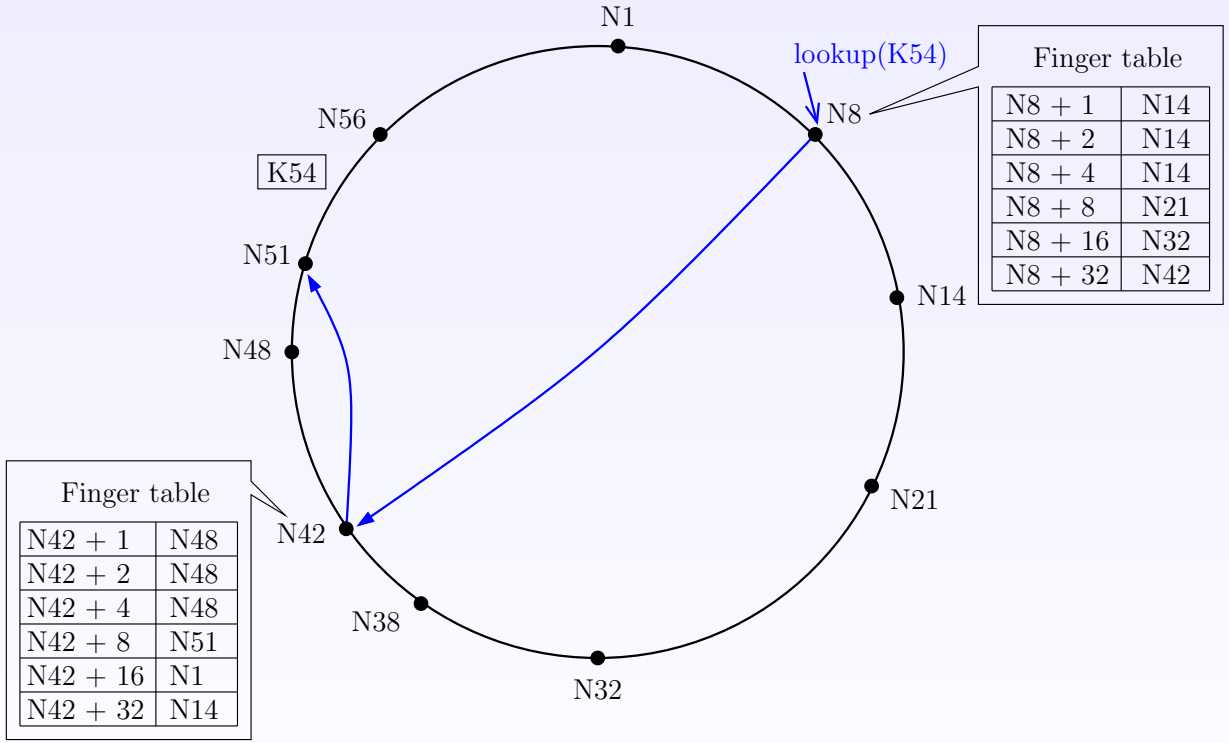
Recherche d'une clé au moyen des *fingers* : exemple



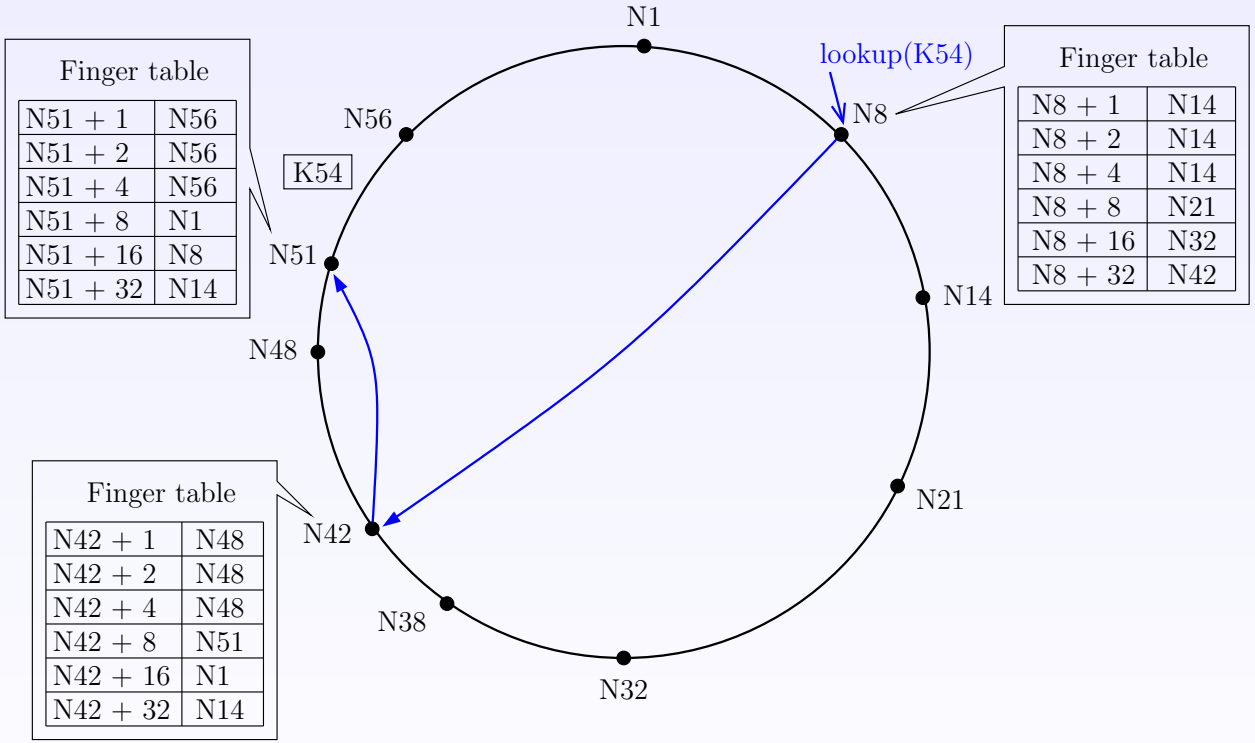
Recherche d'une clé au moyen des *fingers* : exemple



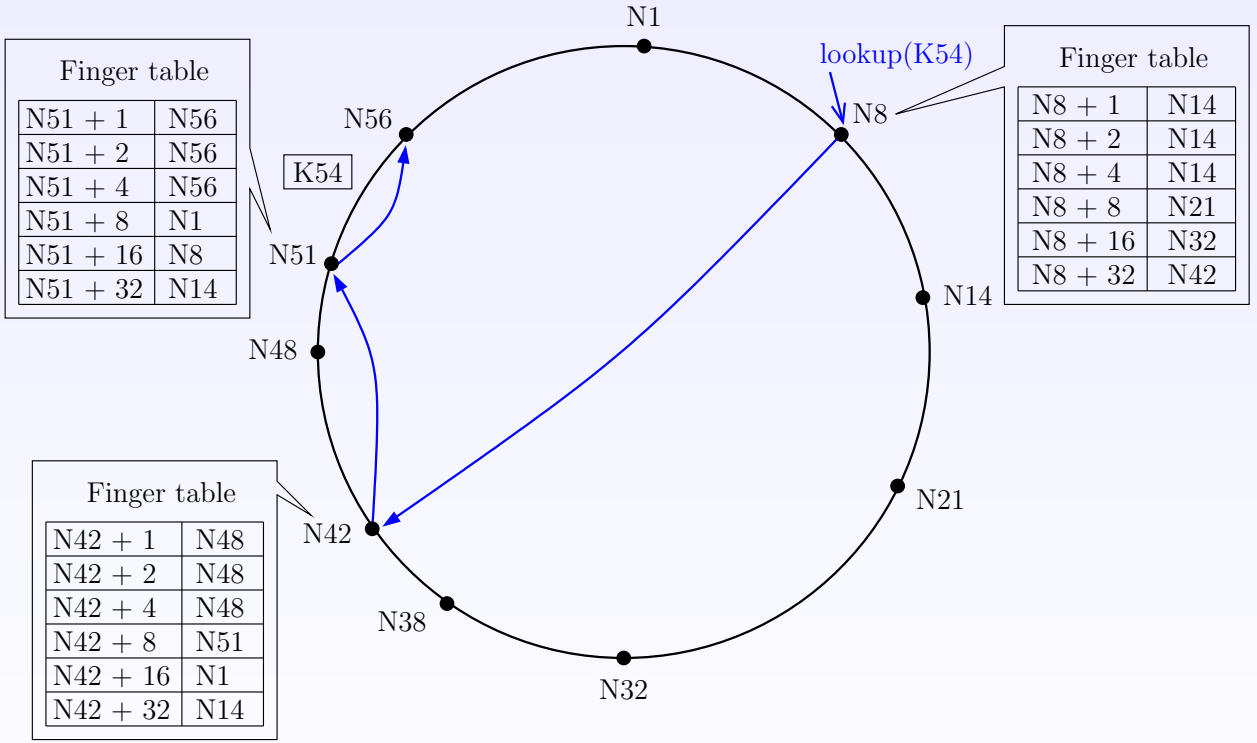
Recherche d'une clé au moyen des *fingers* : exemple



Recherche d'une clé au moyen des *fingers* : exemple



Recherche d'une clé au moyen des *fingers* : exemple



Recherche d'une clé au moyen des *fingers* : algorithme

Le nœud n recherche le successeur de *id* :

$n.\mathbf{find\_successor}(id)$

1: **if** $id \in]n, successor]$ **then**
2: **return** $successor$;
3: **else**
4: $n' = closest\_preceding\_node(id)$;
5: **return** $n'.find\_successor(id)$;

Cherche dans la table locale le plus grand prédécesseur de *id* :

$n.\mathbf{closest\_preceding\_node}(id)$

1: **for** $i = m$ **downto** 1 **do**
2: **if** $finger[i] \in [n, id]$ **then**
3: **return** $finger[i]$;
4: **return** n

Recherche d'une clé : au moyen des *fingers* : propriétés

- Chaque nœud stocke seulement de l'information sur un petit nombre d'autres nœuds.
- L'information stockée concerne principalement des nœuds proches.

Recherche d'une clé : au moyen des *fingers* : propriétés

- Chaque nœud stocke seulement de l'information sur un petit nombre d'autres nœuds.
- L'information stockée concerne principalement des nœuds proches.

Théorème

Avec une forte probabilité, il faut contacter $O(\log N)$ *nœuds pour trouver un successeur dans un réseau contenant* N *nœuds.*

Recherche d'une clé : au moyen des *fingers* : propriétés

- Chaque nœud stocke seulement de l'information sur un petit nombre d'autres nœuds.
- L'information stockée concerne principalement des nœuds proches.

Théorème

Avec une forte probabilité, il faut contacter $O(\log N)$ nœuds pour trouver un successeur dans un réseau contenant N nœuds.

Complexité en pratique : $\approx \frac{1}{2} \log N$.

Ajout et stabilisation (1)

Ajout et stabilisation (1)

Crée un nouveal anneau Chord

$n.\mathbf{create}()$

1: $predecesseur = \mathbf{nil}$;
2: $successeur = n$;

Ajout et stabilisation (1)

Crée un nouveal anneau Chord

$n.\mathbf{create}()$

1: $predecesseur = \mathbf{nil}$;
2: $successeur = n$;

Rejoint l'anneau Chord contenant le nœud n'

$n.\mathbf{join}(n')$

1: $predecesseur = \mathbf{nil}$;
2: $successeur = n'.find\_successeur(n)$;

Ajout et stabilisation (2)

Appelé périodiquement : vérifie le successeur immédiat de n et le notifie.

n.**stabilise**()

1: $x = successeur.predecesseur$;
2: **if** $x \in [n, successeur]$ **then**
3: $successeur = x$;
4: $successeur.notifie(n)$;

Ajout et stabilisation (2)

Appelé périodiquement : vérifie le successeur immédiat de n et le notifie.

$n.\mathbf{stabilise}()$

1: $x = successeur.predecesseur$;
2: **if** $x \in [n, successeur]$ **then**
3: $\quad successeur = x$;
4: $successeur.notifie(n)$;

n' pense qu'il est notre prédécesseur $n.\mathbf{notifie}(n')$

1: **if** $predecesseur \mathbf{isniloun}' \in [predecesseur, n]$ **then**
2: $\quad predecesseur = n'$;

Ajout et stabilisation (3)

Appelé périodiquement. Met à jour la table des *fingers*

next contient le rang du prochain *finger* à mettre à jour

$n.\mathbf{fix\_fingers}()$

1: $next = next + 1$;
2: **if** $next > m$ **then**
3: $next = 1$;
4: $finger[next] = find\_successeur(n + 2^{next-1})$;

Ajout et stabilisation (3)

Appelé périodiquement. Met à jour la table des *fingers*

next contient le rang du prochain *finger* à mettre à jour

n.**fix_fingers**()

1: $next = next + 1$;
2: **if** $next > m$ **then**
3: $next = 1$;
4: $finger[next] = find\_successeur(n + 2^{next-1})$;

Appelé périodiquement. Vérifie si le prédécesseur est fautif.

n.**check_predecesseur**()

1: **if** $predecesseur$ **has failed then**
2: $predecesseur =$ **nil** ;

Correction

- Dès que les pointeurs *successeur* sont à jour, *find_successeur* renverra le bon résultat.

Correction

- Dès que les pointeurs *successeur* sont à jour, *find_successeur* renverra le bon résultat.

Théorème

Si une séquence quelconque d'arrivée de nouveaux nœuds est exécutée en même temps qu'on lieu des opérations de stabilisations, au bout d'un certain temps après la dernière arrivée, les pointeurs successeurs forment un anneau de tous les nœuds dans le système.

Impact des nouveaux nœuds sur les recherches

Comportement possible d'une recherche :

1. Toutes les entrées *fingers* sont raisonablement correctes : succès en temps $O(\log N)$.

Impact des nouveaux nœuds sur les recherches

Comportement possible d'une recherche :

1. Toutes les entrées *fingers* sont raisonablement correctes : succès en temps $O(\log N)$.
2. Tous les pointeurs *successeurs* sont corrects : succès plus lent à obtenir.

Impact des nouveaux nœuds sur les recherches

Comportement possible d'une recherche :

1. Toutes les entrées *fingers* sont raisonablement correctes : succès en temps $O(\log N)$.
2. Tous les pointeurs *successeurs* sont corrects : succès plus lent à obtenir.
3. Les nœuds, dans la région considérée, ont des pointeurs *successeur* incorrects ou les clés n'ont pas encore migré aux nouveaux nœuds : possibilité d'échec.

Impact des nouveaux nœuds sur les recherches

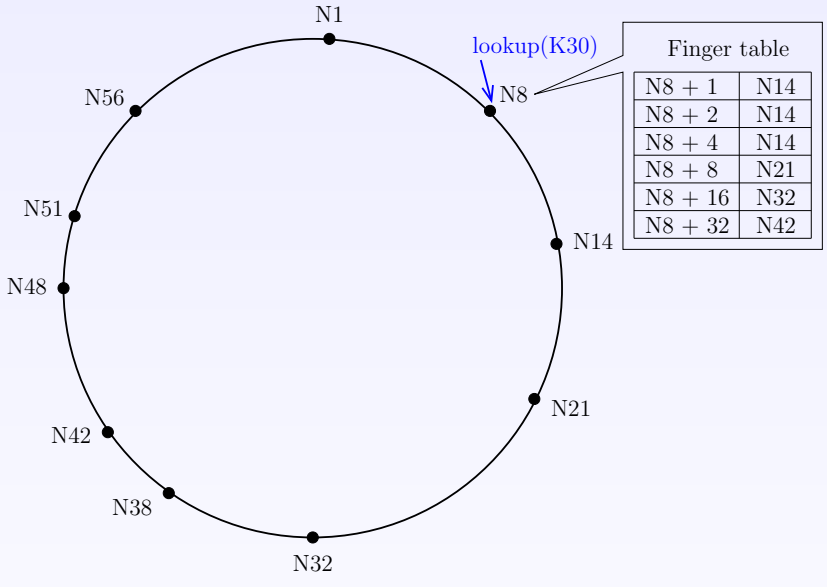
Comportement possible d'une recherche :

1. Toutes les entrées *fingers* sont raisonablement correctes : succès en temps $O(\log N)$.
2. Tous les pointeurs *successeurs* sont corrects : succès plus lent à obtenir.
3. Les nœuds, dans la région considérée, ont des pointeurs *successeur* incorrects ou les clés n'ont pas encore migré aux nouveaux nœuds : possibilité d'échec.

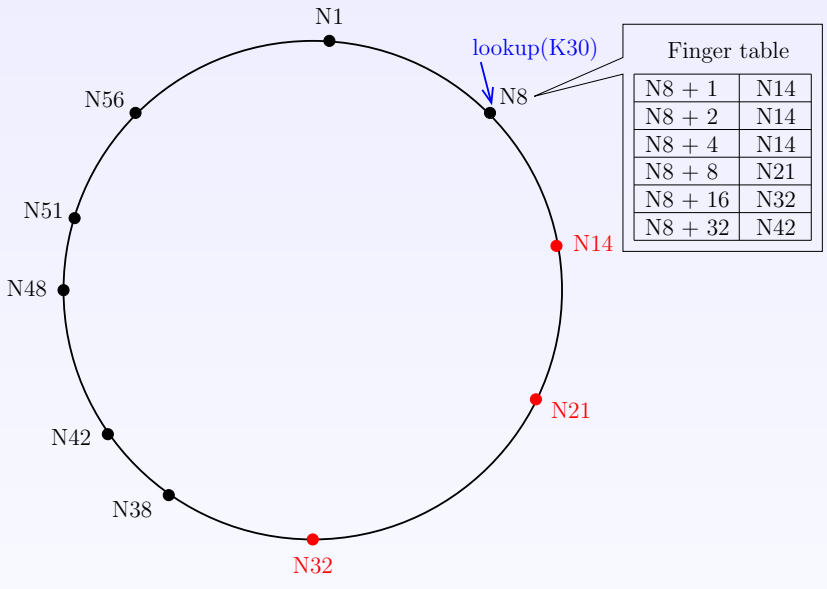
Théorème

Soit un anneau stable à N nœuds dont les fingers sont corrects. Si N nouveaux nœuds rejoignent le système, dès que les pointeurs successeur sont corrects les recherches prennent un temps $O(\log N)$ avec une forte probabilité.

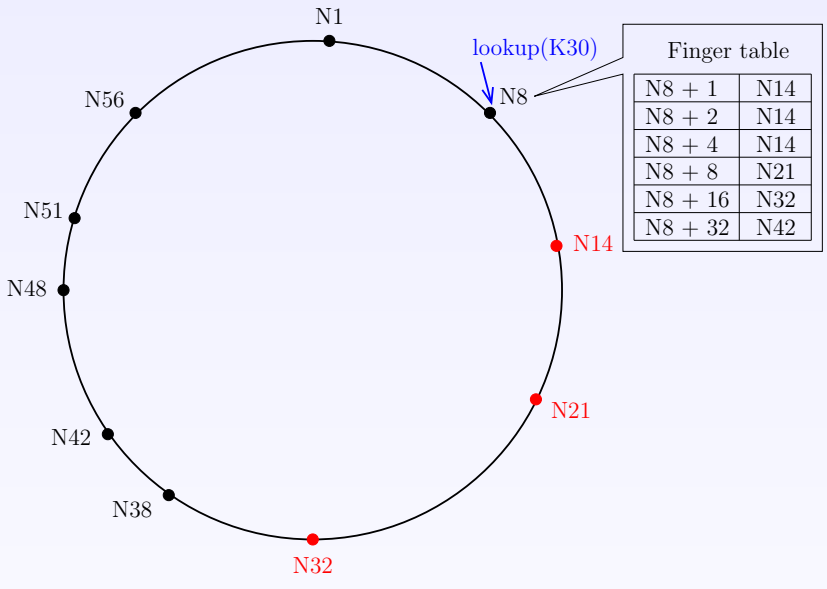
Tolérance aux pannes ?



Tolérance aux pannes ?



Tolérance aux pannes ? Non !



Le nœud 8 va répondre « nœud 42 » au lieu de « nœud 38 ».

Solution

- Chaque nœud maintient une liste de ses r successeurs.

Solution

- Chaque nœud maintient une liste de ses r successeurs.
- Probabilité que les r nœuds soient simultanément en panne : p^r, où p est la probabilité qu'un nœud soit en panne.

Solution

- Chaque nœud maintient une liste de ses r successeurs.
- Probabilité que les r nœuds soient simultanément en panne : p^r, où p est la probabilité qu'un nœud soit en panne.

Théorème

Soit des listes de succeseurs de taille $r = \Omega(\log N)$ dans un système intialement stable. Si chaque nœud tombe en panne avec une probabilité $\frac{1}{2}$, avec une très forte probabilité find_successeur va retourner le plus proche successeur en vie de la clé recherchée.

Solution

- Chaque nœud maintient une liste de ses r successeurs.
- Probabilité que les r nœuds soient simultanément en panne : p^r, où p est la probabilité qu'un nœud soit en panne.

Théorème

Soit des listes de succeseurs de taille $r = \Omega(\log N)$ dans un système intialement stable. Si chaque nœud tombe en panne avec une probabilité $\frac{1}{2}$, avec une très forte probabilité find_successeur va retourner le plus proche successeur en vie de la clé recherchée.

Théorème

Dans un système initialement stable où chaque nœud tombe en panne avec une probabilité $\frac{1}{2}$, le temps d'exécution espéré pour find_successeur est $O(\log N)$.

Départ programmé d'un nœud

- Le système résiste à des pannes simultanées alors un unique départ..

Départ programmé d'un nœud

- Le système résiste à des pannes simultanées alors un unique départ..
- Pour que les choses soient propres :

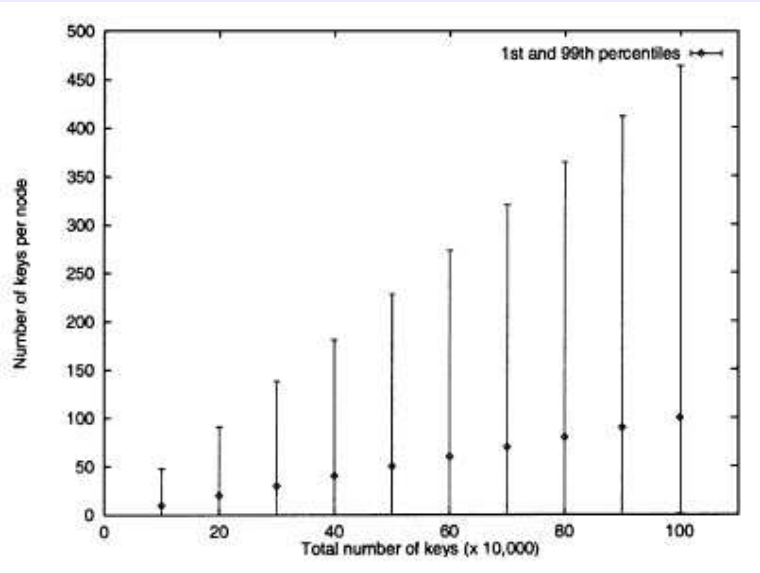
Départ programmé d'un nœud

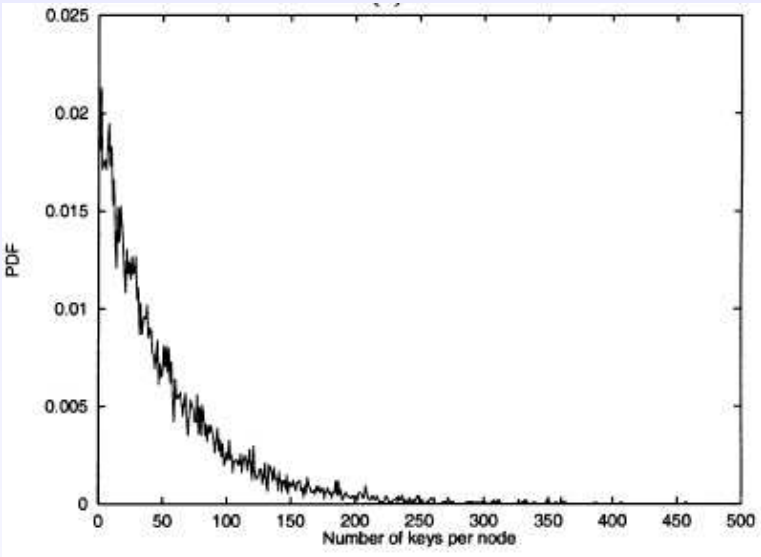
- Le système résiste à des pannes simultanées alors un unique départ..
- Pour que les choses soient propres :
  - Transfert des clés au successeur.

Départ programmé d'un nœud

- Le système résiste à des pannes simultanées alors un unique départ..
- Pour que les choses soient propres :
  - Transfert des clés au successeur.
  - Notification du prédécesseur et du successeur.

Équilibrage de la charge (1)



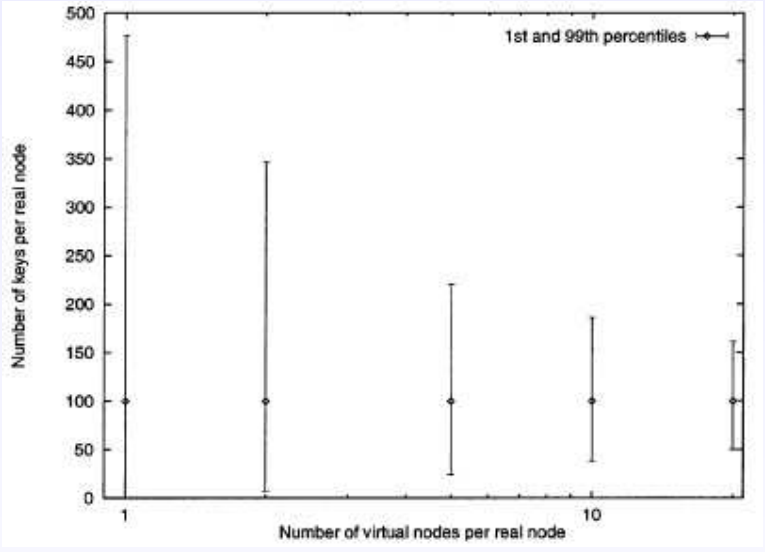


Système de $N = 10^4$ nœuds avec de 10^5 à 10^6 clés.

PDF : probability density function

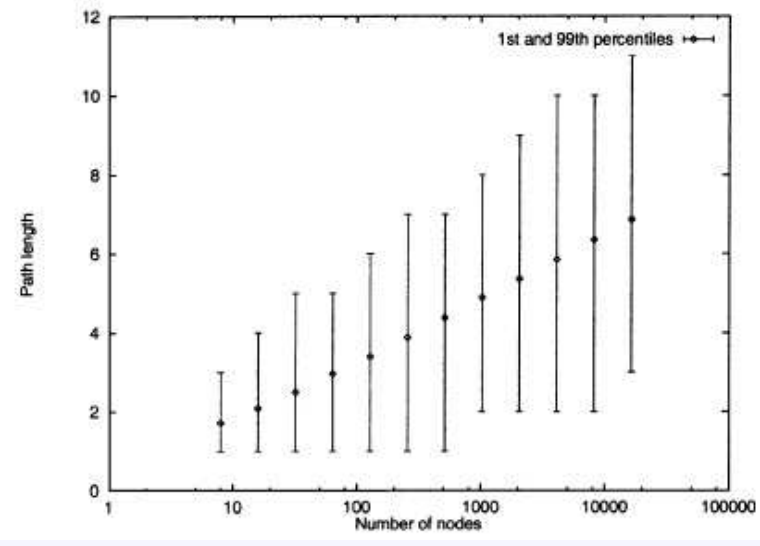
Pour 5.10^5 nœuds, un nœud contient 457 clés, soit 9.1 fois la moyenne.

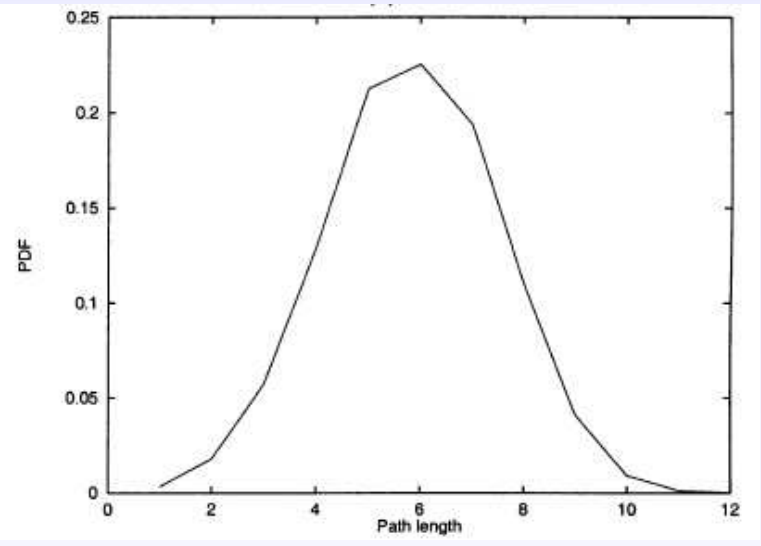
Équilibrage de la charge (2)



On associe $r = 1$, 2, 5, 10 ou 20 nœuds virtuels par nœuds physiques.

Distance de reherche





Nombre de nœuds qui doivent être visités lors d'une recherche

Système de $N = 2^k$ nœuds contenant 100×2^k clés.

k varie de 3 à 14.

Pannes simultanées

Système de 1000 nœuds ; $r = 20 = 2 \log_2 N$.

Une fois que le système est stable, chaque nœud tombe en panne avec une probabilité p.
Ensuite : 10 000 recherches aléatoires.

Fraction of failed nodes	Mean path length (1st, 99th percentiles)	Mean num. of timeouts (1st, 99th percentiles)
0	3.84 (2, 5)	0.0 (0, 0)
0.1	4.03 (2, 6)	0.60 (0, 2)
0.2	4.22 (2, 6)	1.17 (0, 3)
0.3	4.44 (2, 6)	2.02 (0, 5)
0.4	4.69 (2, 7)	3.23 (0, 8)
0.5	5.09 (3, 8)	5.10 (0, 11)

Distance de recherche en présence de pannes.

Recherches pendant la stabilisation

Système de 1000 nœuds ; $r = 20 = 2 \log_2 N$.
Les nœuds rejoignent et quittent le système sans pannes.
La stabilisation est invoquée toutes les 30 secondes.
Fréquence de $R = 0.05$: un nœud rejoint et quitte le système toutes les 20 secondes en moyenne.

Node join/leave rate (per second/per stab. period)	Mean path length (1st, 99th percentiles)	Mean num. of timeouts (1st, 99th percentiles)	Lookup failures (per 10,000 lookups)
0.05 / 1.5	3.90 (1, 9)	0.05 (0, 2)	0
0.10 / 3	3.83 (1, 9)	0.11 (0, 2)	0
0.15 / 4.5	3.84 (1, 9)	0.16 (0, 2)	2
0.20 / 6	3.81 (1, 9)	0.23 (0, 3)	5
0.25 / 7.5	3.83 (1, 9)	0.30 (0, 3)	6
0.30 / 9	3.91 (1, 9)	0.34 (0, 4)	8
0.35 / 10.5	3.94 (1, 10)	0.42 (0, 4)	16
0.40 / 12	4.06 (1, 10)	0.46 (0, 5)	15

Distance de recherche en période de stabilisation.

Plan du cours

Conclusion

- Besoin de structuration des systèmes volatiles.
- Des architectures différentes pour des besoins différents.
- Le succès n'est pas forcément dû à la qualité intrinsue.