Le Pair-à-pair : enjeux, problématiques et quelques solutions

Yves Caniou <yves.caniou@ens-lyon.fr>

Partie du module MIF11 de l'UCBL – Master 1

2005-2006

(version du 15 décembre 2005)

Intérêt du Pair-à-pair

Du routage logiciel

Objectifs et Problématiques :

- Optimiser l'utilisation des ressources
- Passage à l'échelle

Applications

- Partage de fichiers
- Téléphonie
- Multicast

→ Voir les slides d'Olivier !

Première partie

Slides d'Olivier Festor

- Rappels

Ce qui a déjà été vu

- Définitions des notions de Pairs, de systèmes pair-à-pair
- Les cas étudiés :
  - Napster
  - Gnutella
  - Lattice
  - Chord

Deuxième partie

Pastry, Past et Scribe

- Introduction
- PASTRY
  - Design de PASTRY
  - Routage dans PASTRY
  - API
  - Auto-organisation et adaptation
  - Propriétés de localité
  - Résultats
- PAST
  - Introduction
  - Implantation
- SCRIBE
  - Introduction
  - APIs et implantation
  - Gestion des discussions
  - Gestion des groupes
  - Tolérance aux pannes

Sources du cours

Auteurs :

- PASTRY, PAST : Antony Rowstron, Peter Dreschel
- SCRIBE : Miguel Castro, Peter Dreschel, Anne-Marie Kermarrec and Antony Rowstron

Articles de référence pour PASTRY :

- "PASTRY: Scalable, decentralized object location and routing for large-scale peer-to-peer systems"
- "Exploiting network proximity in peer-to-peer overlay networks"

Article de référence pour PAST :

- "PAST : A large-scale, persistent peer-to-peer storage utility"

Article de référence pour SCRIBE :

- "SCRIBE : A large-scale and decentralized publish-suscribe infrastructure"

Contexte

PASTRY : système pair-à-pair complètement décentralisé, passe à l'échelle, fiable.

L'ensemble de mécanismes mis en place dans PASTRY permet la construction d'applications :

- global file sharing, stockage : PAST
- communication de groupe : SCRIBE
- systèmes de nom

Parmis les objectifs :
tenir compte de la localité en minimisant la distance parcourue par les msgs envoyés

Deuxième partie

Pastry, Past et Scribe

Organisation

Pour chaque pair dans l'anneau PASTRY

- NodeId
- Table de routage
- Ensemble de Voisins
- Ensemble de feuilles

Des informations pour mettre en place du routage sur chaque pair : overlay networks !

Les structures de données - 1

NodeId

- codé sur 128bits $\longrightarrow$ permet d'indiquer position dans l'espace circulaire de PASTRY (de 0 à $2^{128} - 1$)
- donné aléatoirement quand le noeud joint : selon une distribution uniforme dans l'espace des possibles (on utilise du hashage crypto sur l'IP address ou sur la clé publique)
- Les nodeId et les clés sont des séquences de digits en base 2^b

Remarque : des noeuds ayant des nodeID adjacents sont distribués géographiquement

Les structures de données - 2

NodeId

Remarque : PASTRY route les msgs vers le noeud dont le nodeID est numériquement le plus proche d'une clé donnée. Comment ?

- ☞ À chaque étape de routage, un pair forwarde à un pair qui partage un préfixe d'au moins 1 digit (c-à-d b bits) de plus avec la clé que lui
- ☞ Si un tel noeud n'existe pas, msg est forwardé au pair avec le même préfixe **mais numériquement plus proche de la clé** que le nodeID du pair actuel.

Exemple :

- $A24398$ et $A24682$ ont même préfixe $A24$
- $FE4561$ est plus proche numériquement de $FE4548$ que $FE600A$

Aucune notion de localité ici !

Les structures de données - 3

Définition : une entrée : couple (nodeID,IPaddress)

Table de routage : R

- En moyenne, $\lceil log_{2^b} N \rceil$[1] lignes de $2^b - 1$ entrées chacune
- Pour une ligne n, $2^b - 1$ entrées représentent chacune
  - un pair avec son nodeID et son IPaddress
  - le nodeID de ce pair et le nodeID du pair actuel ont même préfixe sur n digits
  - le $n+1^e$digit est différent : $2^b - 1$ valeur possibles

Remarque + exemple d'une table de routage (fig. 1) :

- Chaque entrée contient l'adresse IP d'une machine qui a le préfixe donné. En pratique, ce sont en plus les plus proches parmi ceux que le pair connaît
- Choix de b issu d'un compromis entre la taille de la table de routage et le nombre max d'étapes pour router :
  $\lceil log_{2^b} N \rceil * (2^b - 1)$ entrées contre $\lceil log_{2^b} N \rceil$ étapes
  AN :
  - b=4 et N=10^6 pairs, 75 entrées contre 5 étapes de routage
  - b=4 et N=10^9 pairs, 105 entrées contre 7 étapes de routage

[1] dû à la distribution uniforme utilisée

Les structures de données - 4

Ensemble de voisins : M

- Contient |M| entrées des pairs les plus proches (selon métrique)
- Non utilisé pendant routage mais utile pour maintenir propriété de localité
- $|M| \leftarrow 2^b$ ou 2^{b+1}

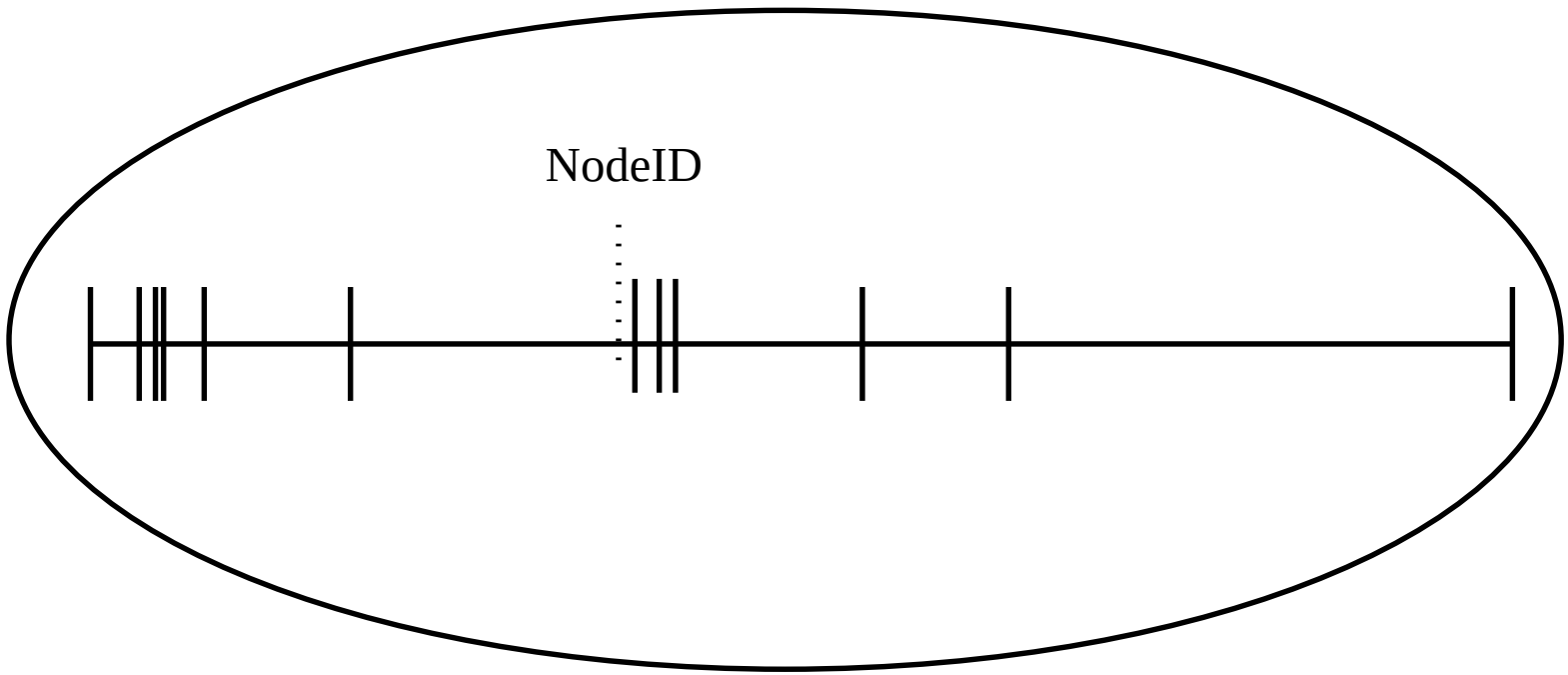
Remarque : la notion de métrique

- un nombre de hop par rapport à routage effectif dans le réseau
- une valeur mesurée par un outil (NWS, IPerf)
- Tous les pairs doivent pouvoir effectuer la mesure !

Les structures de données - 5

Ensemble de feuilles : L

- $|L| \leftarrow 2^b$ ou 2^{b+1}
- Contient $|L|/2$ pairs les plus proches numériquement par valeur inférieure
- Contient $|L|/2$ pairs les plus proches numériquement par valeur supérieure



Deuxième partie

Pastry, Past et Scribe

Notations

- R_{ligne}^{col} avec $0 \leq col < 2^b$ et $0 \leq ligne < \lfloor 128/b \rfloor$: c'est une entrée !
- L_i avec $-\lfloor |L|/2 \rfloor \leq i \leq \lfloor |L|/2 \rfloor$: i^{e}entrée de nodeID le plus proche dans L
- D_l : la valeur des l 1ersdigits de la clé D
- shl(A,B) : longueur du préfixe commun entre A et B, en digits

Algorithme de routage en pseudo-code

Input : La clé D du message. Le pair traversé a le nodeID A

```
if( L_{-⌊|L|/2⌋} ≤ D ≤ L_{⌊|L|/2⌋} ) {
    // D est dans l'intervalle de l'ensemble des feuilles L
    Forward à L_i, de telle sorte que |D − L_i| est minimal ;
}  else {
    // On utilise la table de routage
    l=shl(D,A) ;
    if( R_l^{D_l} ≠ null ) {
        Forward à R_l^{D_l} ;
    }  else {
        // Cas rares où pas d'entrée ou pair associé non joignable
        Forward à T ∈ L ⋃ R ⋃ M, de telle sorte que
            shl(T,D) ≥ l,
            |T − D| < |A − D| ;
    }
}
```

Remarque : La procédure converge toujours puisqu'à chaque étape, il y a routage vers un pair qui partage un préfixe plus grand que le pair actuel ou même préfixe mais plus proche numériquement que le pair local.

Performances

Un nombre d'étapes de routage en $\lceil log_{2^b} N \rceil$ si table à jour et pas de pannes récentes

- Si msg transmis selon R, alors l'ens des pairs avec nodeID dont préfixe convient est réduit par un facteur 2^b à chaque étape
  $\rightarrow$ destination atteinte en $\lceil log_{2^b} N \rceil$
- Si $D \in L$, alors pair destination atteint en au plus une étape
- Si $D \notin L$ et pas d'entrée dans R, alors pair avec préfixe qui convient n'existe pas (dépend de la distribution uniforme, donc de |L|)
  Remarque : si $|L| \leftarrow 2^b$ ou 2^{b+1},
  $p(evt) \leq 0,02$ ou $p(evt) \leq 0,06$ resp.
  $\rightarrow$ 1 étape de routage supplémentaire avec gde proba

Deuxième partie

Pastry, Past et Scribe

API

Fonctions fournies par PASTRY :

- nodeID = pastryInit(Credentials, Application)

  Pour joindre ou créer un nouveau réseau PASTRY ; initialise les états du pair et retourne son nodeID. L'argument *C*redentials est spécifique à l'application et contient les infos utiles pour authentifier un pair. L'argument *A*pplication est un handle vers l'objet dont PASTRY pourra invoquer les méthodes lors de certains évts (arrivée d'un message par ex.)
- route(msg,key)

  Route le message vers le pair dont le nodeID est numériquement plus proche de la clé (parmi les pairs debouts)

Fonctions que doivent implanter les applications utilisant PASTRY :

- deliver(msg,key)

  Appelé qd réception d'un d'un message et que nodeID du pair local est le plus proche numériquement parmi tous ceux debouts
- forward(msg,key,nextID)

  Appelé avant le *f*orward d'un msg vers pair de nodeID=nextID. L'application peut ainsi modifier le msg transmis ou la valeur nextID (si NULL, pas d'envoi de msg)
- newLeafs(leafSet)

  Appelé quand L est modifié afin de permettre à l'application d'ajuster des invariants sur L qui lui sont propres

Deuxième partie

Pastry, Past et Scribe

Ajout d'un pair de nodeID X

Remarque : suppose qu'il connaît un pair A proche de lui

Algorithme

1. X demande le routage de la requête "Join" de clé X
   Remarque : routage jusqu'à pair Z, dont le nodeID est le plus proche num. de X
2. En réponse, les pairs traversés envoient leur table d'état à X à partir desquelles il va construire la sienne (+ demandes possibles d'autres infos de X). Comment ?
   - Comme A supposé proche de X, l'ens. des voisins M_A sera utilisé pour init. M_X
   - Z est pair dt nodeID le plus proche num., donc L_Z utilisé pour construire L_X
   - Table de routage R :
     2.1 R_0 entrées de ${}_AR_0$ car indép. du nodeID
     2.2 ${}_AR_{ligne>0}$ pas utiles pour ${}_XR$ car ne partage pas préfixe
     2.3 Si requête de X routée vers B, B et X partagent préfixe ! alors ${}_BR_1 \rightarrow_X R_1$
     2.4 Si C, alors ${}_CR_2 \rightarrow_X R_2$, etc. pour D, E...
3. X transmet une copie de son état à tous pair de $M \bigcup L \bigcup R$
   qui vont mettre à jour leurs infos (eux mêmes transmettent mise-à-jour etc.)
   $\rightarrow$ géré avec des estampilles insérées dans msgs (cf. Lamport)

Coût total insertion : $O(log_{2^b} N)$ avec $cte \sim 3 * 2^b$

Départ/panne d'un pair de nodeID X

Remarque : suppose panne d'un pair X si ses voisins (dans ξ^{nodeID}, $\in L$) ne peuvent plus le joindre

Algorithme

- Pour remplacer un pair dans L, son voisin contacte pair avec le plus grand index du côté où le voisin est mort, et demande son L
  Exemple : si L_i tombe, avec $\lfloor |l|/2 \rfloor < i < 0$, requête sur $L_{-\lfloor |L|<2 \rfloor}$
  Remarque : Réparation possible tant que pas plus de $\lfloor |L|/2 \rfloor$ pairs morts **dans L** en même temps. Grâce à répartition géo, proba ok même pour |L| faible
- Un pair se rend compte de mort d'un pair via table de routage qd besoin de le contacter
  Le message peut être *forwardé* à autre pair (pas de pb de routage) mais doit être remplacé pour préserver intégrité table de routage
  Exemple : pour réparer ${}_X R_l^d$, le pair X contacte le 1er pair référencé par une entrée ${}_X R_l^i$ $(i \neq d)$, et demande ${}_{pair} R_l^d$. Si pas d'entrée qqsoit i, X contacte pair R_{l+1}^i $(i \neq d)$
- Ens des voisins M pas important pour routage, mais doit être mis à jour pour échange d'infos entre pairs proches
  $\rightarrow$ test périodique vers chaque pair de M
  Si panne, pair demande ${}_A M$ pour $A \in {}_X M$ et calcule la distance (selon métrique) par rapport aux nouveaux pairs reçus et met à jour son M

Deuxième partie

Pastry, Past et Scribe

Localité dans les tables de routage - 1

Définition : peu importe la métrique utilisée (nb de hops, bandwidth disponible), elle doit respecter l'inégalité triangulaire

- Pour mettre en place localité, algo "Join" d'un pair est complété par une heuristique dont le but est de choisir les entrées de sorte à conserver de bonnes propriétés locales
- Dans l'algo, on a vu que ${}_X R_i$ init. avec ${}_{pair} R_i$ des pairs traversés lors du routage de "Join"

1er point

On souhaite maintenir la propriété "toutes les entrées font référence à pair proches du pair local" (parmi tous ceux qui ont un préfixe approprié pour être une entrée). Supposons qu'on ait cette prop., montrons qu'on peut la maintenir :

- On suppose que A proche de X (métrique). Comme ${}_A R_0^i$ proches de A, elles sont proches de X (grâce à inég. Δ), donc prop. préservée
- ${}_X R_1 \leftarrow_B R_1$
  Si on a entrées proches de B, c'est moins clair à quel point B est proche de X, donc pareil pour ses entrées !
  Comme les entrées dans les lignes successives sont choisies d'ens. de tailles décroissantes exponentiellement, distance(B,${}_b R_i$) » |A-B|
  Donc ${}_B R_1$ choix raisonnable pour ${}_X R_1$ (fig. 2)

Localité dans les tables de routage - 2

2^e point

X a init. son état plus ou moins approximativement. La qualité doit être la meilleure pour éviter des erreurs en cascade qui pourraient entraîner une mauvaise gestion de la localité par rapport aux routes employées.

$\rightarrow$ X demande l'état à chaque pair $\in R \bigcup M$ puis compare les distances et met à jour R et M en prenant les plus proches

Remarque : La mise à jour de M est très importante ici car permet de maintenir et propager les infos auprès des pairs proches (au sens métrique), **indép. du préfixe du nodeID**

Localité ds routage affecte les routes PASTRY

Remarque : les entrées dans R sont choisies pour être proches du pair actuel (métrique) parmi ceux qui ont le bon préfixe avec nodeID
Donc, **à chaque étape, le msg est envoyé vers la destination (plus proche du nodeID) en transitant par pair dont la distance est moindre (métrique)**

Mais, Minimisation locale $\not\Rightarrow$ minimisation globale !

Même si les résultats montrent que bonnes routes en $g^a l$, 2 constats :

- Un msg routé de A vers B à distance d de A ne peut pas être routé vers pair à une distance d' inférieure à d de A (Algo)
- La distance traversée par msgs à chaque étape croît exponentiellement :
  Si pair choisit dans R_l parmi ensemble de card $N/2^{bl}$, alors les entrées dans ligne>l sont choisies dans ensemble dont la taille décroît exponentiellement. Comme la distribution des nodeID est unif., la distance à l'entrée la plus proche entre chaque étape successive accroît exponentiellement

$\rightarrow$ Un message de peut ré-entrer un espace circulaire de distance d d'un pair qu'il a traversé (fig. 3)

Le pair le plus proche parmi k pairs, et heuristique

Remarque : Les msgs sont routés vers le pair de nodeID le plus proche de la clé, mais en voyageant le moins possible (métrique)
Réplication et routage :

- Des informations sont répliquées sur k pairs PASTRY (proches num. du nodeID)
- Grâce aux propriétés locales de PASTRY, une route peut arriver à l'un des k pairs proches num. de la destination, mais proche du client (métrique). Dans ce cas, il y a moins de charge, latence, etc.
- PASTRY utilise une heuristique fondée sur densité des nodeID dans $\xi^n odeID$ en utilisant infos locales. À l'aide des ses estimations, elle détecte quand un msg passe près d'un ens de k pairs proches num. et change routage :
  $\longrightarrow$ Prend l'adresse la plus proche num. pour avoir le réplicat le plus proche
  Résultats : 75% réussite de le trouver, 91% si 1 ou 2^{e} plus proche

Deuxième partie

Pastry, Past et Scribe

Résultats d'expérimentation

Fig. 4, 5, 6

Deuxième partie

Pastry, Past et Scribe

Objectifs et fonctionnalités

Outil de stockage global à grande échelle P2P

- Reposant sur la technologie internet
- Capable de passer à l'échelle
- Capable d'assurer grande disponibilité
- Persistence des données
- Sécurité

Techno sous-jacente

- Aspect recherche et routage au dessus de PASTRY
  - Requêtes clientes acheminées de façon fiable et efficace
  - Bonne propriété locale (réseau)
  - Résolution automatique des pbs dus aux pannes de pair et leurs ajouts/retraits
- Utilisation de l'aléa :
  - Des pairs distribués pour stocker réplicats
  - Meilleur équilibrage de charge
- Possible utilisation de smartcards pour systèmes à quota (non dvt ici)

Deuxième partie

Pastry, Past et Scribe

Implantation

Remarques sur les fichiers

- Pour chaque fichier, un ID quasi unique déterminé lors de l'insertion
  Utilisation de 160 bits correspondants au hachage crypto du nom + public key du proprio + seed
  Remarque : pour partage un fichier entre plusieurs utilisateurs, on partage l'ID (+ clé pour déchiffrer)
- Fichiers répliqués pour persistance et disponibilité en terme de :
  possesseurs, géographie, administration, connectivité réseau
  + Des copies supplémentaires de fichiers populaires placés en cache pour répartir la charge
- Pas d'effacement de fichier : PAST ne supporte pas cette fonctionalité
  Le possesseur d'un fichier rappelle le stockage, mais rien ne garantit que le fichier n'est plus disponible

À chaque insertion

- Un certificat du fichier est généré contenant fileID, et signé par proprio
- Coef de réplication k
- Seed
- Date d'insertion
- hash crypto du contenu du fichier

Deuxième partie

Pastry, Past et Scribe

Objectifs et fonctionnalités

Outil de publication/souscription à grande échelle P2P

- Reposant sur la technologie internet
- Capable de passer à l'échelle
- Gestion de groupes de discussions

Techno sous-jacentes : les mêmes que pour PAST !

- Aspect recherche et routage au dessus de PASTRY
  - Requêtes clientes acheminées de façon fiable et efficace
  - Bonne propriété locale (réseau)
  - Résolution automatique des pbs dus aux pannes de pair et leurs ajouts/retraits
- Utilisation de l'aléa

Multicast géré de façon logicielle et complètement décentralisé
et non plus au niveau IP avec des protocoles comme `SRM` (Strong Reliable Multicast) ou `RMTP` (Reliable Message Transport Protocol)

→ Pourquoi implanter du multicast lorsque les routeurs peuvent le faire ?

Deuxième partie

Pastry, Past et Scribe

API de PASTRY/SCRIBE - 1

Slide de rappel, déjà vu en plus détaillé ici

En plus des fonctions suivantes données par l'API de PASTRY

- `nodeID=pastryInit(Credentials)`
- `route(msg,key)`
- `send(msg,IPaddr)`

Il faut aussi définir les fonctions applicatives SCRIBE appelées par PASTRY

- `deliver(msg,key)`
  Appelé qd réception d'un d'un message et que nodeID du pair local est le plus proche numériquement parmi tous ceux debouts
- `forward(msg,key,nextID)`
  Appelé avant le *f*orward d'un msg vers pair de nodeID=nextID. L'application peut ainsi modifier le msg transmis ou la valeur nextID (si NULL, pas d'envoi de msg)
- `newLeafs(leafSet)`
  Appelé quand L est modifié afin de permettre à l'application d'ajuster des invariants sur L qui lui sont propres

API de PASTRY/SCRIBE - 2

Input :

- `topics` est l'ensemble des discussions gérées actuellement
- `msg.source` est le nodeID du pair source du message
- `msg.topic` est l'ID de la discussion ; `msg.type` est le type du message

forward(msg,key,nextID)

```
switch msg.type is
  SUBSCRIBE : if !(msg.topic ∈ topics)
                 topics = topics ⋃ msg.topic
                 route(msg,msg.topic)
              topics[msg.topic].children ⋃ msg.source
              nextID = null // stop routing the orignal message
```

deliver(msg,key)

```
switch msg.type is
  CREATE :      topics = topics ⋃ msg.topic
  SUBSCRIBE :   topics[msg.topic].children ⋃ msg.source
  PUBLISH :     ∀ node in topics[msg.topic].children
                   send(msg,node)
                if subscribedTo(msg.topic)
                   invokeEventHandler(msg.topic,msg)
  UNSUBSCRIBE : topics[msg.topic].children = topics[msg.topic].children - msg.topic
                if( |topics[msg.topic].children| == 0)
                   send(msg,topics[msg.topic].parent)
```

API de SCRIBE

SCRIBE définit en outre ses propres fonctions pour gérer les discussions

- `create(credentials,topicID)`
  Créer une discussion. Les credentials sont utilisés pour des contrôles d'accès
- `subscribe(credentials,topicID,eventhandler)`
  A pour effet d'inscrire le pair local à une discussion d'ID=topicID. Tous les évts reçus pour cette ID sont gérés par le handler
- `unsubscribe(credentials,topicID)`
  A pour effet de désinscrire le pair local
- `publish(credentials,topicID, event)`
  L'évt est publié dans la discussion d'identifiant topicID

Remarque : chaque pair SCRIBE peut publier, être la racine d'un arbre multicast, inscrit à un fil de discussion, un simple nœud dans l'arbre multicast, une éventuelle combinaison de ce qui précède...

Deuxième partie

Pastry, Past et Scribe

Gestion des discussions

Remarques :

- Chaque discussion a un topicID unique
- Le pair dont le nodeID est numériquement le plus proche du topicID gère la discussion

  → C'est le point de rendez-vous pour la discussion, e.g. la racine de l'arbre multicast

Création d'une discussion :

- SCRIBE demande à PASTRY de router un message CREATE dont la clé est `topicID`
- PASTRY route le message vers le pair dont le nodeID est numériquement le plus proche
- La méthode `deliver()` de SCRIBE ajoute la discusion à l'ensemble dont elle a la connaissance
- Elle s'assure que la discussion peut être créer et stocke les crédits correspondants
- Le pair « terminal » devient le point de rendez-vous pour la discussion

Deuxième partie

Pastry, Past et Scribe

Gestion des groupes - inscription

Algorithme d'inscription dans une discussion

- SCRIBE demande à PASTRY de router un message SUBSCRIBE avec une clé `topicID`
- Le message est routé vers le point de rendez-vous
- À chaque pair traversé sur la route, PASTRY invoque la méthode `forward()` définit par SCRIBE, qui au besoin transforme le pair en forwarder et met à jour sa liste de discussion gérée, et ajoute la source comme enfant dans l'arbre multicast
- En devenant un forwarder, il envoie un message SUBSCRIBE au prochain pair (le message de la source est terminé en mettant nextID=null)

Remarques :

- L'arbre multicast est construit avec sa base sur le pair qui est le point de rendez-vous pour la discussion
- Les branches de l'arbres sont déduites d'un mécanisme semblable à RPF, formées par les routes des inscrits vers le point de rendez-vous
- Les nœuds n'ayant que le rôle de faire transiter les messages sont des *forwarders* pour la discussion. Ils peuvent ne même pas être inscrits à la discussion. Ils dressent une table des enfants contenant les entrées (IPaddr,nodeID) des enfants dans l'arbre

Gestion des groupes - désinscription

Algorithme de désinscription dans une discussion

- Le pair local marque la discussion comme non désirée
- **S'il** n'y a pas d'enfant enregistré, il envoie un message UNSUBSCRIPTION à son père dans l'arbre multicast
- Le message parcours récursivement l'arbre et la route est élaguée jusqu'à ce qu'un pair ait encore un enfant pour cette discussion

Remarques :

- Un pair ne connaît le nodeID de son père seulement **après** avoir reçu un évt de lui. Si un nœud voulait se désinscrire sans avoir jamais reçu un évt, l'implantation retarde la désinscription jusqu'à la réception du premier évt.
- L'arbre sera bien équilibré grâce à l'aléa utilisé dans PASTRY
- Les propriétés locales de PASTRY permettent de dire qu'un enfant est plus loin (métrique) que son père (arbre multicast) du point de rendez-vous

Deuxième partie

Pastry, Past et Scribe

- Introduction
- PASTRY
  - Design de PASTRY
  - Routage dans PASTRY
  - API
  - Auto-organisation et adaptation
  - Propriétés de localité
  - Résultats
- PAST
  - Introduction
  - Implantation
- SCRIBE
  - Introduction
  - APIs et implantation
  - Gestion des discussions
  - Gestion des groupes
  - Tolérance aux pannes

Tolérance aux pannes

Pas vu cette année