Structures de données avancées :

Range Partionning: *RP**

Pr ZEGOUR DJAMEL EDDINE
Ecole Supérieure d'Informatique (ESI)
www.zegour.uuuq.com
email: d_zegour@esi.dz

RP*

Famille des SDDS, appelée RP* (Range Partitioning) : RP*N, RP*C et RP*S

- **Fichier RP*N : utilisation exclusive du Multicast. Le client ne comporte aucune image sur le fichier distribué.**
- **Un fichier RP*C : c'est un fichier RP*N avec une image au niveau de chaque client. Utilisation de Unicast et Multicast**
- **Un fichier RP*S : c'est un fichier RP*C + un index distribué au niveau des serveurs indexant toutes les cases. Élimine le multicast.**

RP*N

Structure et Évolution du Fichier

- Un fichier RP*N évolue par éclatement de cases (comme un B-arbre).
- Chaque case (Fichier = ensemble de cases), contient au maximum *b* enregistrements, et est stockée en mémoire centrale sur un serveur.
- Les cases forment une partition sur l'ensemble des clés.
- Chaque case est associée à un intervalle noté (λ , Λ],
  λ : *clé minimale* de la case et Λ : *clé maximale*.

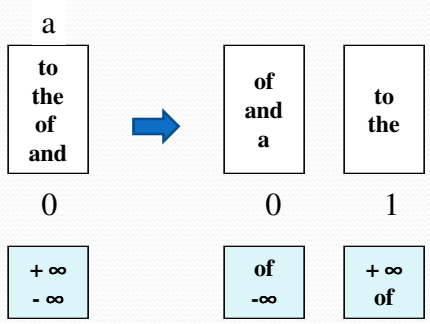
RP*N

Structure et Évolution du Fichier

- Un fichier RP*N est initialement constitué d'une case unique *(case* 0), avec $\lambda = -\infty$ et $\Lambda = +\infty$.
  Toutes les insertions initiales vont à la *case* 0.
  Elle est éclatée lorsqu'elle atteint sa capacité maximale b.
- L'éclatement consiste à déplacer la moitié des enregistrement s vers un nouveau serveur selon une clé du milieu notée *cm*.

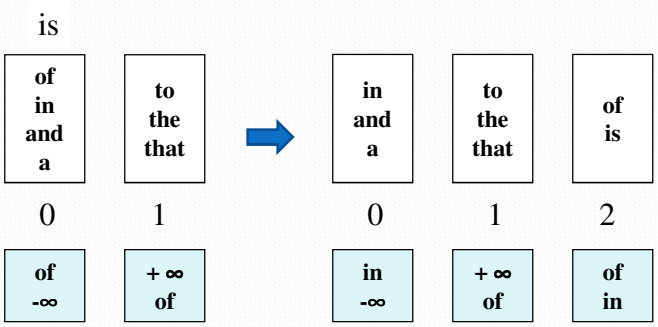
RP*N

Évolution d'un fichier RP*N avec des enregistrements de clé alphanumérique et pour b =4.



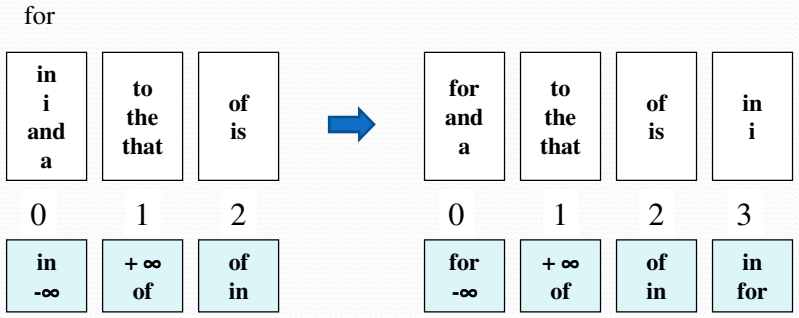
RP*N

Évolution d'un fichier RP*N avec des enregistrements de clé alphanumérique et pour b =4.



RP*N

Évolution d'un fichier RP*N avec des enregistrements de clé alphanumérique et pour b =4.



RP*N

Algorithme d'éclatement d'une case RP*N.

1. Déterminer Cm, la clé du milieu des clés de la case débordée (serveur) A et de la nouvelle clé, cela permet d'obtenir deux groupes *g1* et *g2*.

2. *Allocation d'un serveur S_i*

3. Initialiser l'intervalle de la case du serveur S*i* :

$\lambda(S_i) \leftarrow$ Cm,

$\Lambda(S_i) \leftarrow \Lambda(A)$

4. Déplacer dans S_i l'ensemble des enregistrements de *g2*.

5. Mettre à jour la clé maximale de *A* :

$\Lambda(A) \leftarrow$ Cm

RP*N

Accès au fichier

- Les clients RP*N accèdent au fichier en envoyant des requêtes d'insertion, de mise à jour, de suppression ou de recherche d'enregistrements.

- Ces requêtes sont réparties en plusieurs catégories :
  - ✓ Requête simple : recherche – insertion - suppression - mise à jour d'un enregistrement de clé *c* donnée.
  - ✓ Requête à intervalle : recherche de l'ensemble des enregistrements de clés *c* appartenant à un intervalle donné *[*a, b] *(a* <b)

RP*N

Requête simple

- Coté client :

- Envoyée à l'aide d'un message Multicast.
- Reçue par tous les serveurs.

- Coté serveur :

Chaque serveur S, d'intervalle (λ, Λ], procède comme suit :

✓Si $c \in (\lambda, \Lambda]$ alors *S* exécute la requête, puis envoie éventuellement une réponse au client à l'aide d'un message Unicast.

✓Cette réponse contient le résultat de l'exécution de la requête(par exemple : l'enregistrement de clé *c* pour une recherche avec succès)

✓Si $c \notin (\lambda, \Lambda]$ alors *S* ignore la requête

RP*N

Requête à intervalle

- Il s'agit de la recherche de l'ensemble des enregistrements de clés *c* appartenant à un intervalle donné [a, b] (*a* <b)
- Elle est envoyée à tous les serveurs à l'aide d'un message Multicast.
- Elle est traitée sur chaque serveur d'intervalle (λ, Λ] tel que (λ, Λ] $\cap$ [*a*, b] # {}.
- Les enregistrements sélectionnés sont ensuite envoyés au client en parallèle.

RP*N

Terminaison d'une requête à intervalle

Terminaison déterministe

- Chaque serveur S_i d'intervalle (λ_i, Λ_i] tel que (λ_i, Λ_i] $\cap$ [*a*, b] # {} envoie une réponse au client avec son intervalle (λ_i, Λ_i] et, éventuellement, les enregistrements trouvés.
- Le client termine la requête avec succès, si

$$[a, b] \subseteq \cup (\lambda_i , \Lambda_i]$$

RP*N

Terminaison d'une requête à intervalle

Terminaison probabiliste

- Chaque serveur S_i concerné envoie son intervalle $(\lambda_i, \Lambda_i]$ et, éventuellement, les enregistrements trouvés.
- Le client utilise un *time out t* pour collecter les réponses.
  Ce *time out* est réinitialisé après la réception de chaque réponse.
  La requête est terminée lorsque *t* expire.
- Ce *time out* doit être bien choisi (réduire la probabilité de perte de réponses)
- Ce choix dépend non seulement des performances du réseau mais également de la vitesse de traitement des serveurs.

RP*C

Structure de l'image

- RP*C crée une image du fichier sur chaque client pour réduire l'usage de messages Multicast.
- Image = une collection des intervalles et des adresses des différentes cases déjà accédées.
- Ne reflète pas (en général)la structure exacte de la répartition du fichier sur les serveurs.
- Rôle de l'image:
  - Utiliser un message Unicast si le serveur est déjà référencée.
  - Utiliser un message Multicast dans le cas contraire.

RP*C

Structure de l'image

- Image = une table *T* de couples (A, C), où *A* est l'adresse d'une case et *C* sa clé maximale. (nœud d'un B-arbre)
- Elle est ordonnée suivant les différentes valeurs de C.
- Notations
  A(i) adresse se trouvant à l'entrée i
  C(i) clé à l'entrée i.
  * : adresse inconnue
- Initialement (démarrage d'un nouveau client), *T* ne contient que l'élément (0, ∞].
- Puis, elle évolue en fonctions des *MAI (Messages d'ajustement de l'image)* reçus suite aux erreurs d'adressage.

RP*C

Accès à l'article

(Cas de recherche d'un enregistrement de clé *c (valable pour les* mises à jour et insertions)

<u>Coté Client :</u>

- Désignons par R(c) la requête correspondant à la recherche de la clé c.
- Le client recherche d'abord l'entrée i dans *T* ayant la plus petite clé telle que C(i) ≥c.
- Si A(i) #'*' alors il envoie R(c) par Unicast au serveur d'adresse A(i).
- Sinon R(c) est envoyée à tous les serveurs à l'aide d'un message Multicast.

RP*C

Accès à l'article

Coté serveur :

- Un serveur (recevant R(c)) vérifie d'abord si *c* appartient à son intervalle.

- Cas favorable : le serveur exécute R(c), puis envoie une réponse au client. Cette réponse contient éventuellement l'enregistrement de *c* trouvé, ainsi que l'adresse et l'intervalle du serveur.

- Cas défavorable :
  R(c) est ignorée si elle a été envoyée par Multicast.
  Sinon, il s'agit d'une erreur d'adressage.

- En cas d'erreur d'adressage

✓ Le serveur redirige alors la requête par Multicast vers les autres serveurs.
✓ Le message redirigé contient, en plus de R(c), l'adresse du serveur et son intervalle.

RP*C

Accès à l'article

MAI :

- Toute réponse à une requête envoyée sans erreur d'adressage contient un MAI noté (λ, a, Λ)

✓ L 'intervalle (λ, Λ] est celui du serveur ayant traité la requête
✓ *a* représente son adresse.

- Une réponse à une requête redirigée comporte deux MAI

- Serveur qui a effectué la redirection
- Serveur qui l'a traitée.

- Dans les deux cas, le client corrige son image à l'aide de chaque MAI, selon l'algorithme de mise à jour de l'image.

RP*C

Mise à jour de l'Image

(En entrée : (λ, a, Λ))

1. S'il n'existe pas dans T une entrée *i* de valeur (*a,* C) telle que C(i) = λ et λ #- ∞, alors insérer *(*,* λ) dans T.

2. S'il existe dans T une entrée *i de valeur* (*a,* C) telle C(i) > Λ, alors :

- ✓ **a) Si C(i) =+ ∞ alors Remplacer l'entrée *i par (a,* Λ) puis insérer le couple *(*,* + ∞) dans T.**
- ✓ **b)Sinon, si C(i) <+ ∞ alors Remplacer l'entrée *i par (a,* Λ).**

3. S'il existe dans T une entrée *i de valeur (*,* Λ) alors Remplacer * par *a*

4. S'il n'existe pas dans T une entrée *i de valeur (a,* Λ) alors Insérer *(a,* Λ) dans T.

RP*C

*Évolution de l'image d'un client RP*C.*

MAI

	0 ∞	
(-∞, for, 0)	**0 for | * ∞**	**Règle 2a)**
(in, of, 2)	**0 for | * in | * ∞**	**Règle 1**
	0 for | * in | 2 of | * ∞	**Règle 4**

Table T

RP*C

*Évolution de l'image d'un client RP*C.*

MAI

(of, ∞, 1)	0 for | * in | 2 of | 1∞	Règle 3
(for, in, 3)	0 for | 3 in | 2 of | 1 ∞	Règle 3

Table T

RP*S

Concepts

- RP *S ajoute à RP *C un index réparti au niveau de serveurs spécifiques appelés serveurs d'index.
- Un client RP*S a la même table qu'un client RP*C (Sans '*').
- Cet index représente une image de la structure globale de la répartition du fichier distribué.
- Il est transparent aux clients.
- Objectifs :
  - ✓Éliminer l'usage du Multicast pour envoyer les requêtes simples
  - ✓Accélérer la convergence de l'image du client pour réduire le nombre d'erreur d'adressage.

RP*S

Structure de l'Index Réparti

- L'index RP *S = structure semblable à celle d'un fichier en arbre-B+ distribué.

- Deux types de serveurs :

- Les serveurs d'index correspondent aux nœuds non terminaux de l'arbre

- Les serveurs de données représentent les feuilles.

RP*S

Structure de l'Index Réparti

- Chaque serveur d'index peut contenir au maximum *m* (*m* >>1) couples (A, C) ordonnés.

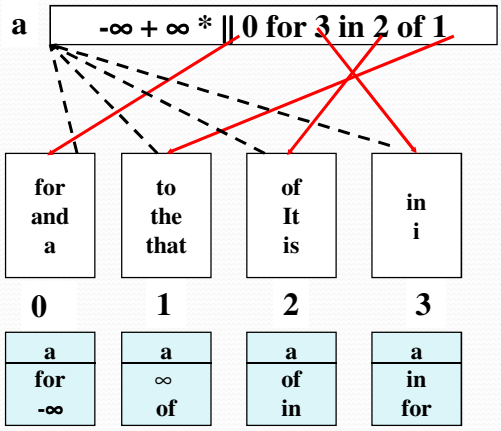
- Serveur d'index = nœud d'un B-arbre

Avec un en-tête contenant son intervalle et un pointeur vers son nœud parent (utilisé pour la redirection en cas d'erreur d'adressage)

- Chaque case (serveur) de données est muni d'un en-tête contenant son intervalle et un pointeur vers le nœud feuille d'index le plus à gauche (même remarque)

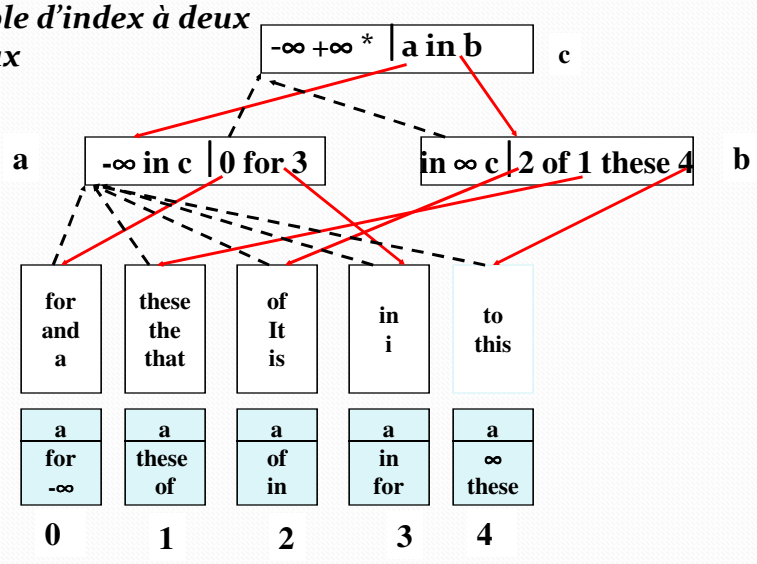
RP*S

Exemple d'index à un niveau



RP*S

Exemple d'index à deux niveaux



RP*S

Évolution du fichier

1. La case *0* est éclatée comme dans RP *N. Il en résulte la création du premier nœud *a*

a

-∞ + ∞ ‖ * ‖ 0 Cm_1 1

2. Le nœud a est créé avec le triplet *(0, Cm_1, 1)*. *Cm_1* : clé du milieu.

a

-∞ + ∞ ‖ * ‖ 0 Cm_1 1 Cm_2 2 ….

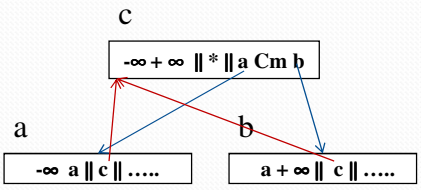
3. Tout autre éclatement entraîne l'insertion d'un nouveau couple *(Cm_r,r)* dans a

RP*S

Évolution du fichier

4.Quand le nœud a devient plein, il est divisé en 2

5. Le nouveau nœud racine *c* *est créé avec l'*intervalle *(- ∞,+ ∞)* et de pointeur père *. C'est le père des nœuds *a* et b.



6. Etc.

RP*S

Accès au fichier

(Algorithme de traitement d'une requête simple dans RP *S)

Coté Client :

Recherche dans son image l'adresse du serveur.

Coté serveur :

Soit *S* le serveur possédant la case d'intervalle (λ , Λ] recevant R(c)

1. Si *c* ∈ (λ , Λ], alors *S* exécute la requête.

2. Sinon, *S* redirige la requête vers son nœud père P.

3. Si c ∈ (λ *p*, Λ p] de P, *P* alors envoie la requête à la case contenant l'enregistrement de clé c.

4. Sinon, P redirige la requête au niveau supérieur.

5. Ce processus est répété jusqu'à ce qu'une feuille ou la racine soit atteinte.

6. La feuille finale renvoie un MAI au client avec la trace du chemin parcouru par la requête dans l'arbre.

RP*S

Accès au fichier

(Algorithme de traitement d'une requête simple dans RP *S)

- MAI = (Adresse du serveur, Clé maximale dans ce serveur)
- La structure représentant le chemin parcouru est appelée Arbre-AI (Arbre de l'ajustement de l'image).
  C'est ensemble des MAI.
- Suivant l'état de son image, le client peut envoyer directement une requête à un serveur d'index au lieu d'un serveur de données. (considéré comme une erreur d'adressage)
- Dans ce cas, le serveur d'index redirige la requête vers la bonne branche de l'arbre distribué, pour quelle soit acheminée vers le serveur correcte.

RP*S

*Algorithme de mise à jour de l'image d'un client RP *S.*

Soit *T* l'image du client (comme dans RP *C)

T = { (*a, s*), *a* =serveur , s=clé }.

Cet algorithme est exécuté pour chaque nœud n de l'*arbre-AI et pour chaque (a, s) de n* de bas en haut et niveau par niveau.

RP*S

*Algorithme de mise à jour de l'image d'un client RP *S.*

Pour chaque nœud feuille *n (index) de l'arbre AI* et pour chaque couple (a,s) ∈ n, modifier *T* uniquement dans les cas suivants :

Si *s* ∉T, ajouter (a, s) *à T* en respectant l'ordre de rangement.

Sinon,
si ∃(a, s') ∈ *T* avec *s'*#s, alors *s'* ← *s* [Modification de la borne supérieure de l'intervalle d'une case déjà enregistrée dans l'image]

Si non, si ∃(*a'*, s) ∈ *T* avec *a'* # a, alors *a'* ← *a* [Modification de l'adresse d'une case déjà enregistrée dans l'image]
→ *Transforme les adresses de nœuds en adresses de cases.*

RP*S

Algorithme de mise à jour de l'image d'un client RP *S.

Pour chaque (a, s) d'un nœud non feuille n de l'arbre-AI, modifier *T* uniquement dans les cas suivants :

Si *s* ∉ T, alors ajouter (a, s) *à T*

Si ∃(*a'*,s) ∈ *T* avec *a'* # *a* et *s* = Λ (n) avec Λ (n) # ∞ , alors :

Soient s' le prédécesseur de *s* dans n, s'' le prédécesseur de *s* dans *T*

Si s'' ≤ s', alors *a'* ← a.

RP* : Conclusion

- **Facteur de chargement : le même pour les 3 méthodes(70% pour les insertions aléatoires)**
- **En terme de messages :**
  Rp*N meilleur que Rp*C meilleur que Rp*S
- **En terme de convergence des images des clients**
  Rp*S meilleur que Rp*C
- **Rp*s : schéma déterministe ordonné(fonctionne sans multicast)**
  Inconvénient : exige beaucoup de message lors des opérations de recherche, insertion, ajustement, etc.