Partage et gestion de données distribuées

Master 2 Informatique - UFR S.A.T

Pr. Ousmane THIARE

othiare@ugb.edu.sn
[www.ousmanethiare.com]

13 septembre 2025

Partage et gestion de données distribuées

Introduction

Comment faire pour qu'une information soit vue par plusieurs sites comme si elle était présente localement sur chacun des sites et manipulée par des processus s'exécutant sur ce site SACHANT qu'éventuellement une copie de cette information se trouve sur chacun des sites. D'où deux problèmes principaux :

- nommage et disponibilité de la donnée
- cohérence de celle-ci

Désignation

Nommage

Le nom d'un objet est formé d'un part, de son nom proprement dit et d'autre part, de la voie d'accès à cet objet $\Longrightarrow$ dépend du niveau d'abstraction.

Ainsi, par exemple, en UNIX, un fichier a 4 "noms" :

- nom symbolique local ou global
- descripteur dans une application
- descripteur en mémoire lors de l'utilisation
- inode

Un **nom interne** est une information désignant des objets et destinée à l'usage interne du système :

- pour une machine : adresse internet ou ethernet, ...
- pour des unités d'informations : inodes, numéros de pages, ...
- pour des moyens de communication : des numéros de ports, ...

Désignation

Nommage

La **désignation ou nommage** est la fonction qui permet :

- d'associer des noms aux objets, c'est-à-dire d'associer un nom symbolique à un nom interne interprétable par les couches basses du système $\Longrightarrow$ phase de création du nom
- d'interpréter un nom pour retrouver l'objet, c'est-à-dire de retrouver son nom interne $\Longrightarrow$ phase d'utilisation.

Ce nom doit :

- désigner l'objet de manière unique permettre de retrouver l'objet être difficile à contrefaire

Désignation

Nommage
Exemple : pour l'unicité :

- on peut ajouter le numéro de série de la machine à tout nom
- sur le réseau internet, on rajoute le l'adresse internet pour la sécurité, on simule les capacités par l'utilisation de fonctions de cryptage des champs

MAIS problème lors du déplacement (migration) de l'objet.

Migration et nom interne

Si un objet ne migre que très rarement on peut inclure dans le nom une information sur la localisation physique de l'objet. Par contre, pour les objets "mobiles", les noms doivent être "invariants" et donc ne pas dépendre de la localisation de l'objet.
Si l'objet est peu mobile :

- solution "simple" : prévenir les utilisateurs (problème : qui sont-ils ?)
- utilisation de liens de poursuite à partir de son site d'origine :
  - utilisation de liens de poursuite à partir de son site d'origine problème de nombre de sites à visiter avant de le retrouver problème de la rupture de la chaîne en cas de panne ou d'arrêt d'un site problème de la durée de vie et la validité de ces liens

Migration et nom interne

Si l'objet est très mobile :

- soit utilisation d'une base de données centralisée sur un serveur
  - problème du goulet d'étranglement
  - problème en cas de panne du serveur problème de la validité de la base : si un objet bouge plus vite que la base n'est mise à jour

  soit utilisation de la diffusion
  - problème de la saturation du réseau
  - problème de l'accessibilité en diffusion

Recherche de noms

Comment trouver un nom interne à partir d'un nom symbolique ?

- manuellement
- utilisation de fichiers :/etc/hosts ;/etc/passwd...mais alors problème de la connaissance"initiale"et de l'évolution sur chaque machine
- serveur de désignation (NIS, NIS+, ...)
  tous les fichiers précédents sur les noms sont "remplacés" par des fichiers sur un seul site. **Exemple :** tous les utilisateurs sont déclarés sur une machine (exple : ada) qui stocke leur identification (nom, home, etc..) et leur mot de passe. Les machines clientes peuvent l'interroger pour accepter ou non une connexion. Ainsi, si le problème de la connaissance initiale n'est pas résolue, la mise à jour des informations est simplifiée MAIS encore, faut-il savoir que les données doivent être mises à jour ! par exple, si une

Recherche de noms

- serveur de noms (DNS) : utilisé principalement pour les adresses internet.
  Tous les couples (nom symbolique, adresse internet) d'un réseau (exple : ugb.sn) sont connues par une machine serveur (exple : isis -> 130.79.200.1). Cette machine peut être interrogée par toutes les machines du monde. Ainsi si, je cherche à connaître l'adresse internet de la machine dollar.big-money.com, j'émets une requête vers le serveur du mon réseau (mail.ugb.sn) qui ne connaît pas cette adresse. Il transmet donc au serveur du réseau .sn (à St-Louis). Celui-ci transmet la demande au serveur du réseau .com qui lui la transmet au serveur du réseau big-money.com. (Il peut y avoir une étape de plus .sn -> NIC -> .com). Enfin, ce serveur donne l'adresse qui refait le chemin inverse => il suffit donc que chaque machine du réseau soit déclaré au

Désignation symbolique et système de fichiers

Le problème principal dans l'union de deux sous-systèmes est que chacun a déjà son propre espace de noms.

Exemple : comment unifier deux arborescences UNIX ?

- création d'une super racine virtuelle (cas de UNIX United).
  Pour un utilisateur connecté sur M2, le fichier A2 a pour nom A2. Par contre pour un utilisateur connecté sur M1, ce fichier a pour nom /../M2/A2 => le nom dépend de la machine distante
- montage logique (cas de NFS)

NFS est un ensemble de logiciels permettant de partager des fichiers sur un réseau hétérogène (Unix, PC, Mac...) sur un modèle serveur/client. Toute machine peut être client et celles disposant d'un disque peuvent être en plus serveurs. Il est construit sur XDR et RPC et utilise UDP. Il permet une désignation transparente à l'utilisateur

Désignation symbolique et système de fichiers

Principes de fonctionnement : Une machine désirant mettre son disque (ou une partie de son disque) à la disposition de clients exporte une partie de son arborescence qui peut alors être montée comme un répertoire local par les clients. On parle alors de répertoire distant. Ainsi si on monte le répertoire /D1/A2 d'une machine M2 sur le répertoire /A2 d'une machine M1 alors, le fichier f1 se nomme /A2/f1 sur M1 et /D1/A2/f1 sur M2. Mais dans les deux cas, l'utilisateur ne connaît pas le nom "réel" et la localisation du fichier f1. (ce pourrait être le répertoire A2 de M1 qui est monté sur /D1/A2 de M2).

Désignation symbolique et système de fichiers

Principes de fonctionnement : Attention :
NFS est un système à serveurs sans état. Un serveur NFS ne conserve pas d'information sur l'état de ses clients => une panne d'un client n'a pas d'effet sur le serveur (par contre, un client est mis en attente lors d'une panne du serveur) car toute l'information sur la position, sur l'état des fichier (ouvert, mode ...), etc. , est dans les clients (les requêtes sont auto-suffisantes) alors que dans un système à états tel que AFS, des tables permettent de connaître les fichiers ouverts et par qui, la position dans ces fichiers pour chacun des clients, etc.

Désignation symbolique et système de fichiers

Principes de fonctionnement : Avantages :

Serveurs sans états (ou mémoire)	Serveurs à états (ou mémoire)

tolérance aux pannes	messages plus petits
Open et close facultatifs	meilleures performances
pas d'espace sur le serveur pour tables	lecture anticipée possible
nombre de fichiers ouverts illimié	idempotence plus simple
pas de problème en cas de panne client	possibilité de vérouiller un fichier

Désignation symbolique et système de fichiers

Principes de fonctionnement : Le fait que NFS soit sans état a aussi pour conséquence que lorsqu'un serveur modifie un fichier, il ne peut prévenir ses clients. Comme chaque client comporte un cache-disque, il se pose donc une problème de cohérence et de mise à jour.
$\Longrightarrow$ Dans NFS, chaque écriture est transmise immédiatement au serveur. Un client vérifie périodiquement son cache : si une information y est depuis plus de 3 secondes, il considère qu'elle n'est plus valable. Cela a pour conséquence, une augmentation du nombre de requêtes, de mise à jour, ..., d'où une augmentation du trafic réseau, d'où une limitation du nombre de clients pouvant être servis (max de 10 à 20). Et en plus le cohérence n'est pas assurée.

Désignation symbolique et système de fichiers

Réalisation : Le noyau doit être configuré pour pouvoir participer au partage de fichiers (tous, ou presque tous les systèmes le sont par défaut) : il y a modification du système des inodes. Les fonctions noyau concernées par des opérations d'E/S sur les fichiers décodent différemment les inodes en fonction du type du fichier. Cette information leur sont fournies grâce à l'appel de la fonction fss (file system switch). On parle alors de VINOD (virtual inod).

Désignation symbolique et système de fichiers

Administration et démons : Le serveur

Commande	Fichiers paramètres par défaut	Commentaires
exportfs, share	/etc/exports, /etc/dfs/dfstab	exporte un répertoire
showmount		visualise les montages
nfsd		démon pour les requetes NFS
mountd		démon pour les montages NFS

Le client

Commande	Fichiers paramètres par défaut	Commentaires
mount	/etc/fstab	montages des répertoires distant
biod		démon assurant la gestion du cache local

Désignation symbolique et système de fichiers

Contrôle des accès :

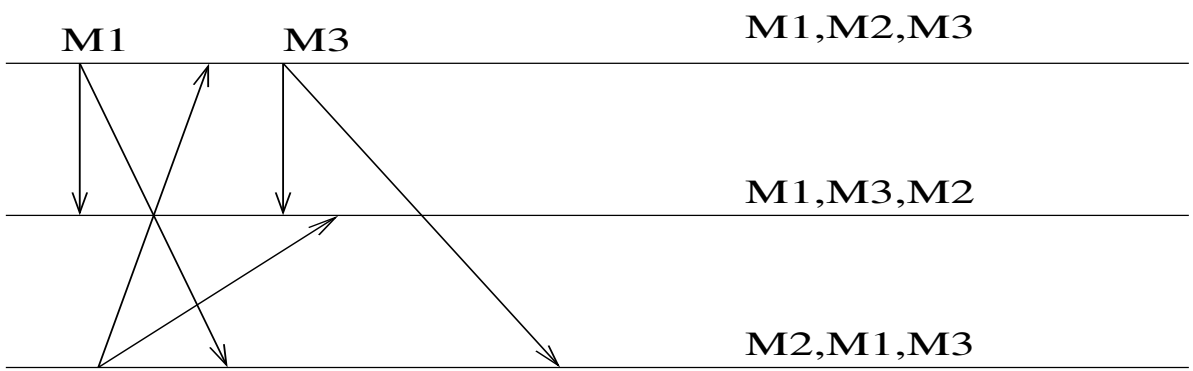
- le contrôle de la correspondance entre l'utilisateur et le propriétaire se fait par l'uid mais NFS n'unifie pas les uid sur les différentes machines. De plus, si un uid ne correspond à rien sur une machine distante, NFS lui affecte l'uid de l'utilisateur nobody. Idem pour root.
- en NFS, à chaque accès à un fichier, les droits sont vérifiés. Néanmoins, le propriétaire conserve les droits initiaux même s'il les modifie.
- Le serveur NFS autorise les lectures sur un fichier binaire exécutable même si le droit de lecture n'est pas donne ? : il ne sait pas si le client lit le fichier ou s'il est en train de le charger.

Cohérence (consistance) de données dupliquées

La cohérence est un problème surtout lorsque l'utilisateur ou l'application font des copies explicites d'informations ou lorsque le système ou l'application font des copies "cachées" de fichiers d'informations par utilisation de caches.

On définit trois niveaux de cohérence :

- *cohérence faible :* la mise à jour de toute copie doit avoir lieu au bout d'un temps fini.
  - l'ordre des mise à jour d'une information n'est pas nécessairement celui de ses modifications
  - une copie peut être temporairement incohérente

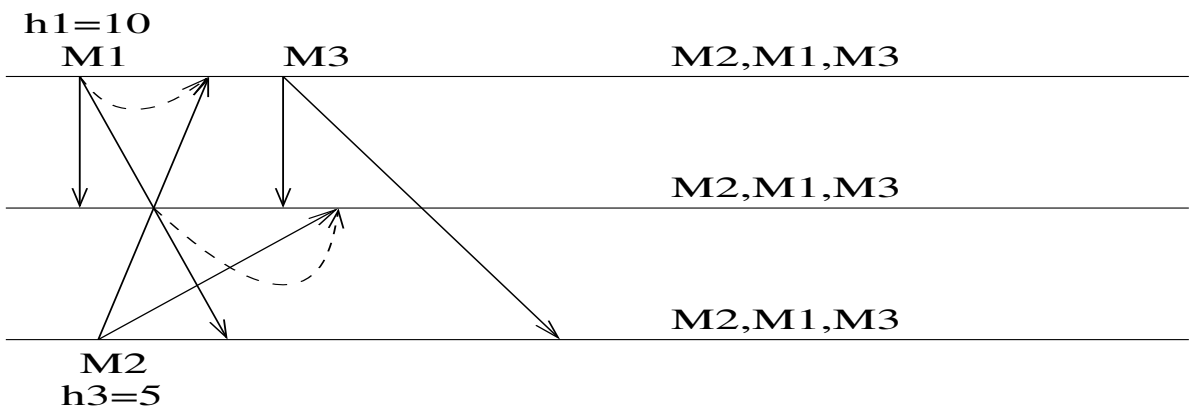


Cohérence (consistance) de données dupliquées

L'ordre de prise en compte des modifications Mi n'est pas la même sur tous les sites. Par contre, à un moment, la donnée doit être mise à jour : dépend de l'application. Il est évident que les opérations Mi doivent être "commutatives".

Cohérence (consistance) de données dupliquées

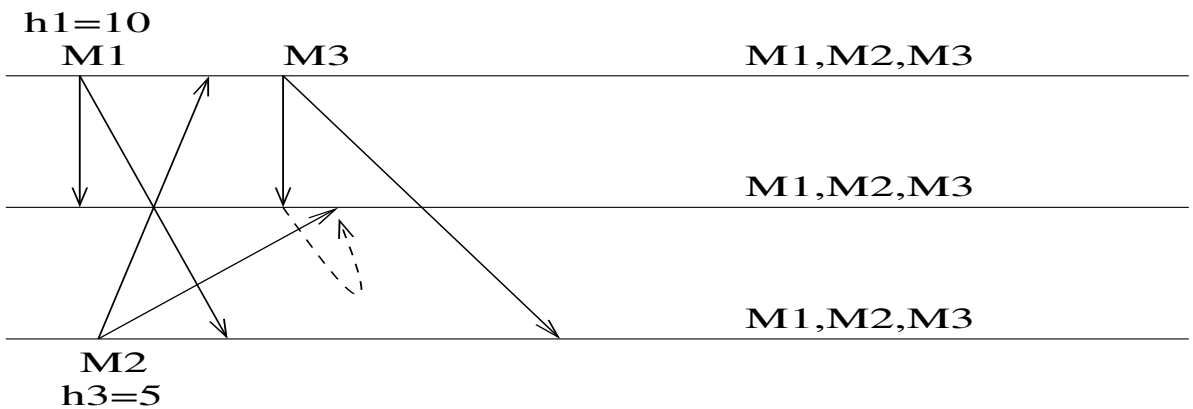
- *cohérence forte :* toute consultation doit refléter le résultat de toutes les modifications antérieures une copie lorsqu'on la consulte, est toujours à jour.
  Ainsi dans l'exemple précédent, S1 va d'abord s'assurer que sa modification est faite APRES la modification M2. Puis seulement, il l'envoie.



De même, S2 fera les modifications dans l'ordre en "retardant" éventuellement les messages.

Cohérence (consistance) de données dupliquées

- *cohérence causale :* si deux événements qui causent des modifications peuvent être globalement ordonnés alors les modifications correspondantes doivent être faites dans le même ordre. Ainsi, dans l'exemple précédent, les modifications M2 et M1 sont indépendantes donc peuvent être traitées dans n'importe quel ordre. Par contre, M2 est cause de M3 , elle doit donc être faite avant.



D'où, S2 retarde M3. Deux aspects à étudier : maintien de la cohérence et détection de l'incohérence

Cohérence (consistance) de données dupliquées

On trouve deux "types" de caches : des caches mémoire (qui contiennent des données image de données de la mémoire centrale) et des caches fichiers (qui contiennent des fichiers ou des blocs de fichiers image de fichiers disque) Suivant ces types, les protocoles de maintenance de la cohérence sont différents. Il existe trois approches permettant de vérifier et valider des données dans un cache :

- vérification déclenchée par le client (cas des systèmes de fichiers répartis) : le client déclenche un contrôle soit à chaque accès à une donnée, soit à l'ouverture des fichiers uniquement ou enfin, périodiquement, à intervalle de temps fixe.
- vérification déclenchée par le serveur : le serveur enregistre pour chaque client les (parties de) fichiers que les clients mettent en caches. Quand il détecte une incohérence potentielle, il réagit

Cohérence (consistance) de données dupliquées

- vérification par invalidation et espionnage (cas des caches mémoire) : dans ce cas , l'architecture la plus utilisée (et en fait, la seule vraiment réalisable) est celle fortement couplée avec un bus unique : Chaque fois qu'une donnée est écrite dans un cache, elle est aussi écrite en mémoire centrale (cache du type write-through cache). Les lectures réussies n'engendrent pas de trafic. Tous les caches surveillent constamment le bus. Chaque fois qu'un cache détecte une écriture à une adresse mémoire dont il possède le contenu, il agit (par exemple, suivant le protocole MESI). On parle alors de cache espion (snoopy cache). Ceci garantit la cohérence des caches.

Cohérence (consistance) de données dupliquées

Principales solutions :

- le site primaire est utilisé : il donne ou non l'autorisation de modification d'une donnée et assure la diffusion si nécessaire
- le site modificateur demande d'abord à tous les autres sites l'autorisation de modifier
- un jeton circule : il permet l'utilisation et modification de la donnée. Il contient de plus, la liste des modifications successives.

Ces trois solutions s'apparentent à une forme d'exclusion mutuelle. Elles garantissent un cohérence forte de la donnée. Mais, elles sont lourdes à mettre en place et interdissent en fait des modifications "simultanées".

Cohérence (consistance) de données dupliquées

Principales solutions :

Or, dans certains cas, des modifications d'une donnée (ajout de données dans un fichier, ...) peuvent être faites indépendamment sur plusieurs sites, la seule contrainte est que l'ordre dans lequel ces modifications sont prises en compte soit le même sur tous les sites. D'où l'idée de dater toutes les modifications de façon globale (horloge logique ou vectorielle) et de les réaliser sur tous les sites dans cet ordre.

Cohérence (consistance) de données dupliquées

Un algorithme de maintien de la cohérence “à la demande”

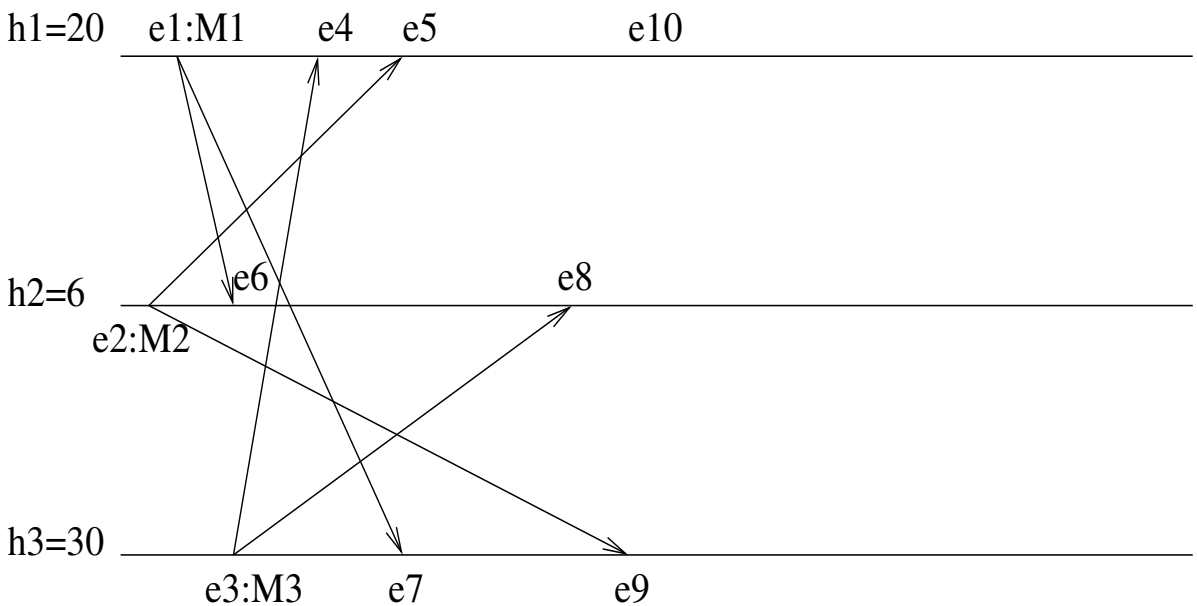
Idée : Lorsqu’une modification est faite sur un site, les autres sites ne sont pas nécessairement obligés d’en tenir compte immédiatement. On va donc s’arranger pour que tous les sites soient au courant des modifications, mais que chacun décide lui-même du moment opportun de les traiter : soit parce qu’il a du temps, soit parce qu’il a besoin de connaître l’état de la donnée pour continuer, ou autre raison. On espère ainsi limiter le nombre de message dans le système.

Principe : Pour remettre à jour une donnée, un site doit réaliser toutes les modifications antérieures dues aux autres sites. On va donc maintenir un datage des événements de modifications.

Cohérence (consistance) de données dupliquées

Un algorithme de maintien de la cohérence “à la demande”

On suppose que le réseau est FIFO. Sur l’exemple suivant



Cohérence (consistance) de données dupliquées

Un algorithme de maintien de la cohérence "à la demande"

Remarque : Sans algorithme, S1 traiterai M1 puis M2 puis M3 ; S2 traiterai M2 puis M1 puis M3 ; S3 traiterai M3 puis M1 puis M2

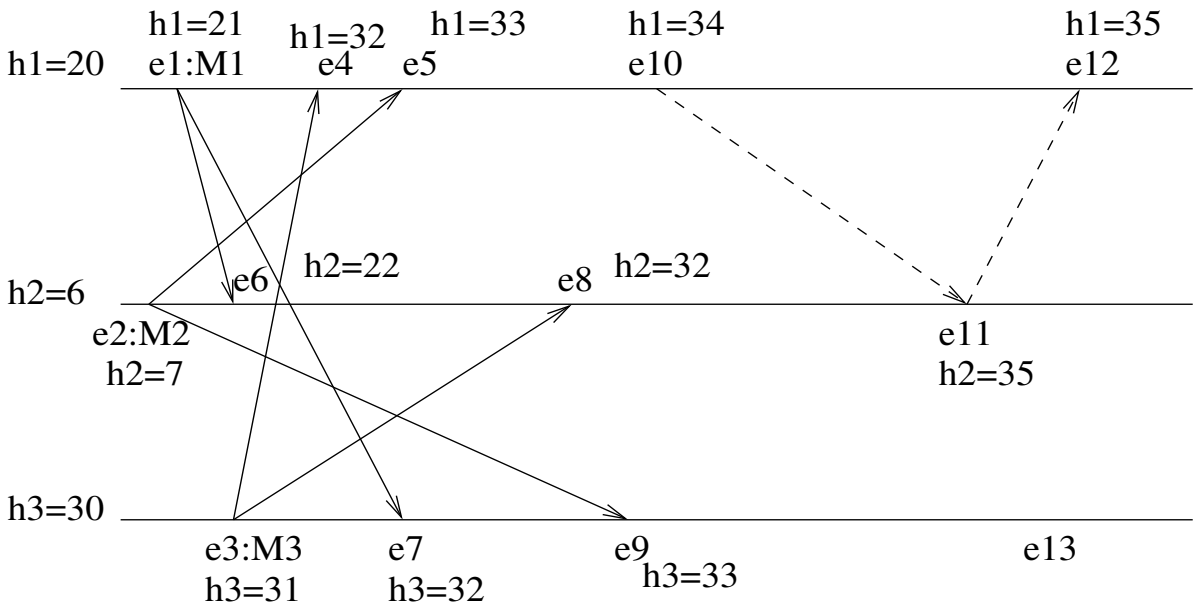
Supposons qu'en e10, S1 veuille prendre en compte sa demande de modification M1.

En e4, grâce au message M3, il sait qu'il n'a pas de demande de modification provenant de S3 de date inférieure à la date de M1car date(M3) = (31, 3). Il recale son horloge h1 = (31,s 22) + 1 = 32. Par contre en e5, il sait que date(M2) = (7, 2) < date(M1). Il recale son horloge h1 = (7, 32) + 1 = 33. Il doit traiter M2 en premier. Il ne peut toujours pas traiter M1 car rien ne lui dit que S2 n'a pas refait une demande en (8,2). S1 va donc faire une "enquête".

Cohérence (consistance) de données dupliquées

Un algorithme de maintien de la cohérence “à la demande”

En e10, S1 envoie E1(33, 1) à S2. Puis h1 + +.



En e11, S2 va recaler son horloge h2 = max(date(E1) = 33, h2 = 32) + 1= 34 et répondre par un acquittement A1 daté par (34, 2) puis incrémenter son horloge h2 = 35.

En e13, S3 devra faire une enquête auprès de S1 et S2.

Cohérence (consistance) de données dupliquées

Un algorithme de maintien de la cohérence "à la demande"

L'algorithme : Chaque site Si dispose d'un tableau F tel que F[j] contient la liste des messages reçus par Si en provenance de Sj.

Lorsque Si veut faire modification, il la date et l'émet en diffusion.

Puis, lorsque Si veut prendre en compte les modifications, il doit exécuter toutes les modifications reçues et ce suivant leur date de modification. Pour cela, il parcours les listes contenues dans F tant qu'elles contiennent des modifications de date inférieure à la date de la modification qu'il veut prendre en compte OU qu'elles soient vides. Mais lorsqu'une liste est vide, rien ne garantit qu'une modification n'est pas en train de se faire sur un autre site ou qu'un message annonçant un modification n'est pas en train d'arriver : le site Si doit

Cohérence (consistance) de données dupliquées

Un algorithme de maintien de la cohérence "à la demande"

L'algorithme :

Demande_de_modif(Mk){
Hi = Hi + 1 ;
dater Mk par Hk=Hi
diffuser Mk(Hi, i)
inserer Mk(Hk, i) dans F[i]
}

Prise_en_compte(Mk(Hk, i)){
Faire {
Vider_liste((Hk, i))
V =$\{j \text{ tel que } F[j] = \emptyset\}$
$\forall j \in V$ envoyer E(Hi,i) à Sj
Attendre au moins un acquittement ou autre message de chaque site $Sj \in V$

Cohérence (consistance) de données dupliquées

Un algorithme de maintien de la cohérence “à la demande”
L’algorithme :
Reception Enquete(h, j)
{ Hi =max(h,Hi)+1
envoyer A(Hi,i) à Sj
}

Cohérence (consistance) de données dupliquées

Un algorithme de maintien de la cohérence “à la demande”

L’algorithme :

Vider_liste(Mk(Hk , i)){

Faire {

$W = \{j / estampille\ de\ tete(F[j] < (Hk, i)\}$

Si Type de tete(F[j]) = Modif {

executer la modification

}

Enlever (tete(F [j])) }

tantque $W \neq \emptyset$

}

Cohérence (consistance) de données dupliquées

Un algorithme de maintien de la cohérence “à la demande”
L’algorithme :
Reception Modif(M,h,j) {
Hi =max(h,Hi)+1
insérer M(h,j) dans F[j] }

Cohérence (consistance) de données dupliquées

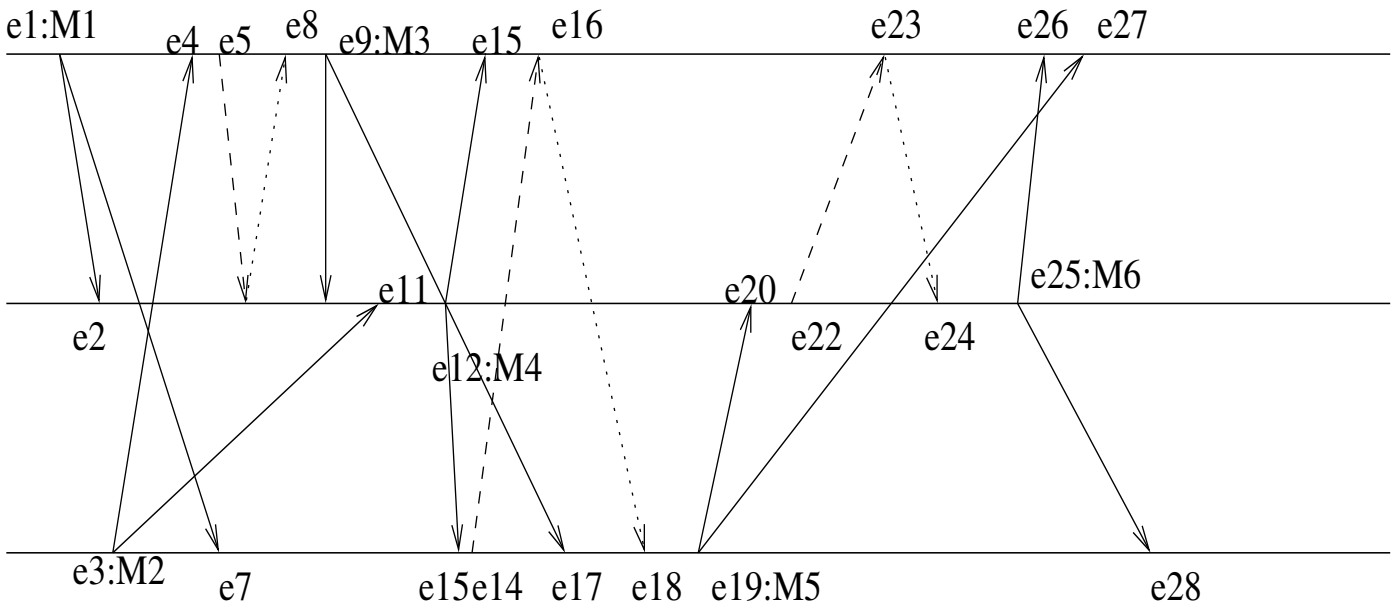
Un algorithme de maintien de la cohérence “à la demande”
L’algorithme :
Reception Acquittement(h,j) {
Hi =max(h,Hi)+1
insérer A(h,j) dans F[j]}

Cohérence (consistance) de données dupliquées

Un algorithme de maintien de la cohérence “à la demande”

L’algorithme :

Un exemple complet :



Voyons ce que cela donne (A VERIFIER) :

Cohérence (consistance) de données dupliquées

L'algorithme :

	e1	e2	e3	e4	e5	e6	e7
H1	1			2			
F[1]	M1(1,1)			M1(1,1)			
F[2]				ens vide			
F[3]				M2(1,3)			
Action	M1(1,1)				E(2,1)––S2		
H2		2				3	
F[1]		M1(1,1)					
F[2]							
F[3]							
Action						A(3,2) S1	
H3			1				2
F[1]							M1(1,1)
F[2]							
F[3]			M2(1,3)				M2(1,3)
Action			M2(1,3)				

En e5, P1décide de prendre en compte sa modification M1. Il sait qu'il ne peut plus avoir de demande de modification inférieure à la sienne provenant de P3. Par contre il doit engager pour P2

Cohérence (consistance) de données dupliquées

	e8	e9	e10	e11	e12	e13
H1	4	5				
F[1]	M1(1,1)	M3(5,1)				
F[2]	A(3,2)	A(3,2)				
F[3]	M2(1,3)	M2(1,3)				
Action	Traiter M1					
H2			6	7	8	
F[1]			M1(1,1)M3(5,1)	M1(1,1)M3(5,1)	M1(1,1)M3(5,1)	
F[2]					M4(8,2)	
F[3]				M2(1,3)	M2(1,3)	
Action					M4(8,2)	
H3						9
F[1]						M1(1,1)
F[2]						M4(8,2)
F[3]						M2(1,3)
Action						Traiter M1 (1,1)

En e13, alors, P3 aurait pu décider de ne pas prendre en compte la modification M1. Mais, il avait un peu de temps, l'a fait.

Cohérence (consistance) de données dupliquées

Puis, dans la foulée, il décide de prendre en compte les autres modifications : e14

	e14	e15	e16	e17	e18	e19
H1		9	10			
F[1]		M3(5,1)	M3(5,1)			
F[2]		M4(8,2)	M4(8,2)			
F[3]		M2(1,3)	M2(1,3)			
Action			A(10,1)−−−S3			
H2						
F[1]						
F[2]						
F[3]						
Action						
H3	9			11	12	13
F[1]	∅			M3(5,1)	M3(5,1), A(10,1)	M3(5,1), A(10,1)
F[2]	M4(8,2)			M4(8,2)	M4(8,2)	M4(8,2)
F[3]	M2(1,3)			M2(1,3)	∅	M5(13,3)
Action	E(9,3) −− S1			Traiter M2(1,3)		M5(13,3)

L'algorithme :

	e20	e22	e23	e24
H1			16	
F[1]			M3(5,1)	
F[2]			M4(8,2)	
F[3]			M2(1,3)	
Action			A(16,1)––S2	
H2	14	15		17
F[1]	M1(1,1), M3(5,1)	∅		A(16,1)
F[2]	M4(8,2)	M4(8,2)		M4(8,2)
F[3]	M2(1,3), M5(13,3)	M5(13,3)		M5(13,3)
Action	Traiter M1, M2, M3	E(15,2) ––– S1		Traiter M4(8,2)
H3				
F[1]				
F[2]				
F[3]				
Action				

Cohérence (consistance) de données dupliquées

L'algorithme :

	e25	e26	e27	e28	e...
H1		20	21		
F[1]		M3(5,1)	M3(5,1)		∅
F[2]		M4(8,2), M6(19,2)	M4(8,2), M6(19,2)		∅
F[3]		M2(1,3)	M2(1,3), M5(13,3)		∅
Action					Traiter M2,M3,M4...
H2	19				
F[1]	A(16,1)				
F[2]	M6(19,2)				
F[3]	A(16,1)				
Action	M6(19,2)				Traiter M6
H3				20	
F[1]				M3(5,1), A(16,1)	A(16,1)
F[2]				M4(8,2), M6(19,2)	
F[3]				M5(13,3)	
Action					Traiter M2,M3,M4...

Cohérence (consistance) de données dupliquées

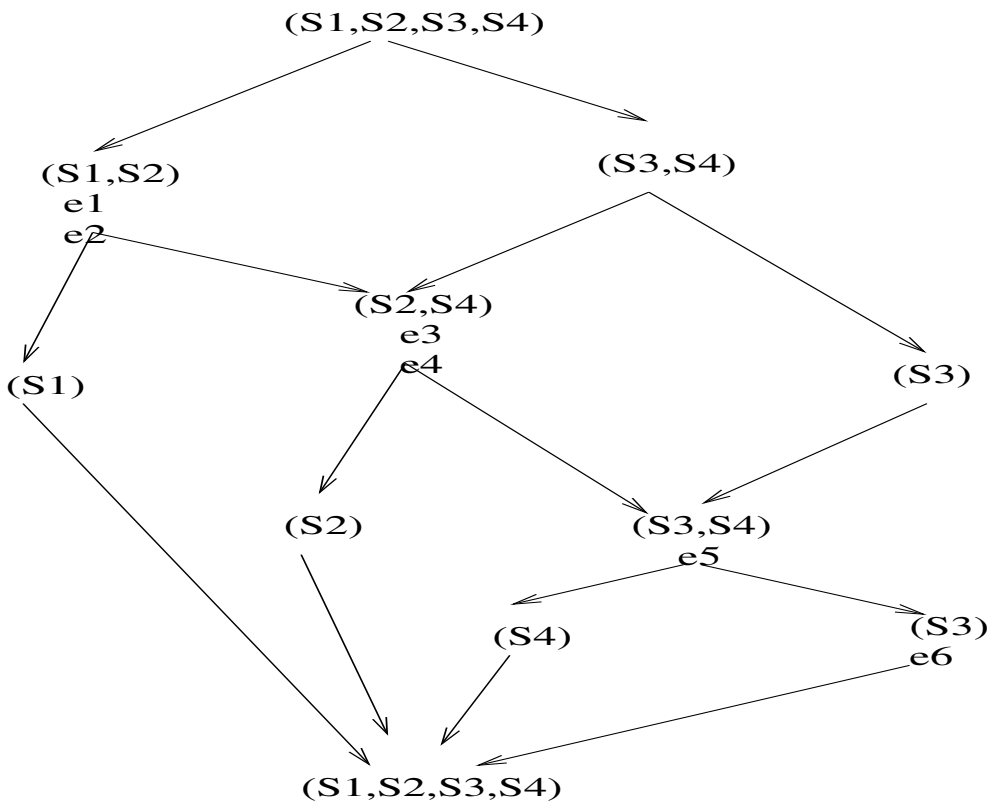
Position du problème : On considère un réseau constitué de plusieurs sites dont chacun dispose d'une copie d'une information. Deux cas se présentent :

- il n'y a aucun protocole de maintien de la cohérence => il n'y a tout simplement rien à espérer : pour voir si des copies d'une donnée sont incohérentes, il suffit (gasp) de comparer toutes les copies.
- Tous les sites suivent un même protocole qui assure la cohérence des copies il peut quand même y avoir incohérence des copies . Ainsi, dans le cas où le réseau s'est coupé en N sous-réseaux déconnectés. Chaque sous réseau peut maintenir la cohérence "locale". Mais que ce passe-t-il lorsque les sous-réseaux sont reconnectés ?

Cohérence (consistance) de données dupliquées

Détection de l'incohérence mutuelle

On peut représenter par un graphe l'évolution du réseau. Par exemple, un réseau à 4 sites S1, S2, S3 et S4 et l'évolution suivante :



Cohérence (consistance) de données dupliquées

Détection de l'incohérence mutuelle

Problèmes :

- les copies des sous-réseaux qui se reconfigurent pour n'en former plus qu'un sont-elles identiques ou non ?
- dans le cas où elles ne le sont pas, les copies d'un sous-réseau (et qui donc ont la même valeur) sont-elles "plus à jour" que les copies de l'autre sous-réseau : en d'autres termes, une convergence des deux ensembles de copies est-elle possible ou non $\Longrightarrow$ dans ce dernier cas, il y a incohérence.

Cohérence (consistance) de données dupliquées

Détection de l'incohérence mutuelle

Principe de l'algorithme de Parker pour la détection de l'incohérence : Détecter l'identité de deux ensembles de copies et leur incompatibilité éventuelle nécessite de mémoriser l'histoire des ces copies sur chaque site -> on associe à chaque copie, un vecteur qui mémorise cette histoire.

Initialement V = [S1 : 0, S2 : 0 , S3 : 0, S4 : 0]

Ce vecteur indique pour chaque site, le nombre de modifications qu'a effectué le site. Prenons l'exemple des 4 sites précédents. Supposons que S1 ait modifié 1 fois la donnée (événement e1), S2 deux fois (événements e2 et e4), S3 une fois (événement e6) et S4 deux fois (événement e3 et e5).

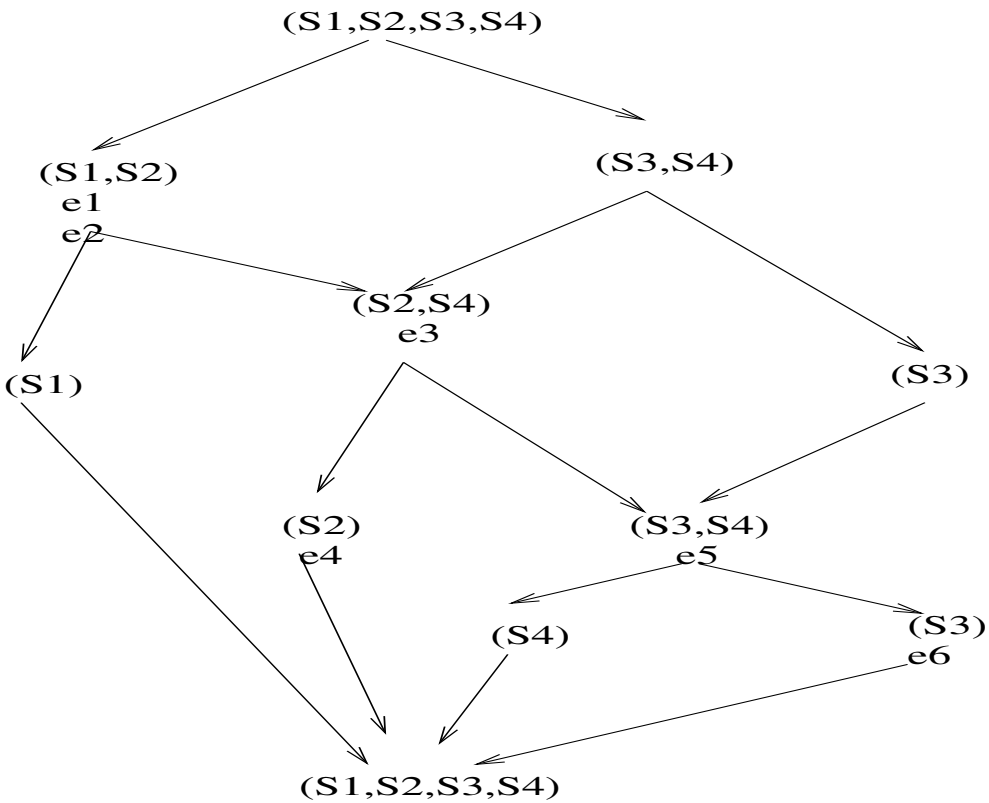
Chaque fois que grâce à l'algorithme de maintien de cohérence, Si prend en compte une modification de la donnée due à Sk, Si incrémente $V_i[k]$.

Sur chaque site Si, on devrait avoir V_i

Principe de l'algorithme de Parker pour la détection de l'incohérence : Mais si le réseau se coupe : des sites ne vont plus recevoir les informations et donc les modifications à apporter à la donnée. Ainsi, par exemple, si dès le début, S2 est coupé des autres sites alors, lorsqu'une modification aura lieu sur S2 , alors ni S1, ni S3 , ni S4 ne seront prévenus.
Dans notre exemple, supposons que, lors de la reconfiguration du réseau, on aura
V1 =[S1 :1,S2 :1,S3 :0,S4 :0] V2=[S1 :1,S2 :2,S3 :0,S4 :1]
V3 =[S1 :1,S2 :2,S3 :1,S4 :2] V4=[S1 :1,S2 :2,S3 :0,S4 :2]
On voit que V3 > V1, V3 > V2 et V3 > V4 : V3 "regroupe" les modifications qu'ont subies les copies $\Longrightarrow$ la copie associée à V3 est plus à jour : on la prend comme donnée finale. Que ce passe-t-il si e4 a lieu après la coupure de (S2, S3, S4) en (S2) ; (S3, S4) ?

Cohérence (consistance) de données dupliquées

Détection de l'incohérence mutuelle



Cohérence (consistance) de données dupliquées

Détection de l'incohérence mutuelle

On aura :
V1 =[S1 :1,S2 :1,S3 :0,S4 :0] V3 =[S1 :1,S2 :1,S3 :1,S4 :2]
V2=[S1 :1,S2 :2,S3 :0,S4 :1] V4 =[S1 :1,S2 :1,S3 :0,S4 :2]
On voit que V2 > V1 les histoires sur S1et S2sont compatibles. Idem pour S3 et S4 car V3 > V4. Par V2 et V3 sont incompatibles $\Longrightarrow$ les copies associées sont issues d'histoires différentes => il y a donc incohérence.
SI on ne trouve pas de Vi supérieur à tous les autres : PERDU $\Longrightarrow$ on peut plus reconstruire la donnée ! !