École Doctorale de Grenoble
Master 2 Recherche "Systèmes et Logiciel"

Tolérance aux fautes-2
Serveurs à haute disponibilité

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/people/krakowia

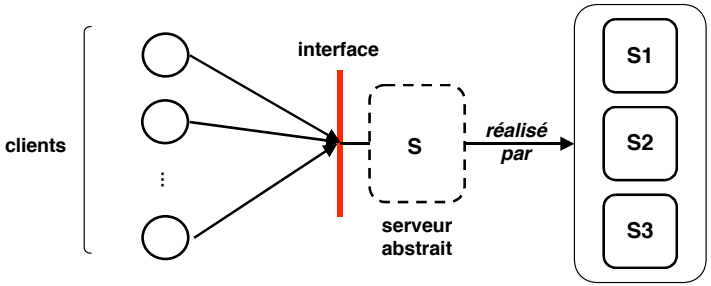
Disponibilité des serveurs

- **Motivation et objectifs**
  - Le modèle client-serveur est un schéma de base de construction d'applications réparties
  - L'objectif est de présenter les principales techniques permettant d'augmenter la disponibilité des serveurs
- **Plan**
  - Conditions de cohérence pour des serveurs dupliqués
  - Techniques de duplication
    - Serveur primaire-serveur de secours
    - Réplication active
  - Outils de base pour la duplication cohérente
  - Exemple d'application

Référence de base : R. Guerraoui, A. Schiper. Software-based Replication for Fault-Tolerant Systems, *IEEE Computer,* vol. 30, 4, April 1997, pp. 68-74

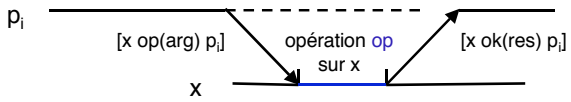
Serveurs tolérants aux fautes : principes

- **La tolérance aux fautes est obtenue par redondance**
  - On utilise N serveurs pour résister à la panne de N-1 serveurs
  - Hypothèse : panne franche (*fail stop*)
- **La redondance impose des contraintes de cohérence**
  - Vu du client, l'ensemble des N serveurs doit se comporter comme un serveur "abstrait" unique (les spécifications d'interface du serveur doivent être respectées)



Un modèle pour les serveurs dupliqués

Un ensemble de processus $\{p_1, \ldots, p_n\}$ interagit avec un ensemble d'objets x, … via des appels synchrones (requête - réponse)



Pour la tolérance aux fautes, chaque objet x existe en plusieurs exemplaires x^1, x^2, … x^N
Comment définir la cohérence de ces exemplaires multiples ?

On définit une relation d'ordre sur les appels. On suppose qu'il existe un système de datation global avec validité forte (soit t) - en pratique horloges physiques synchronisées
Soit $t_{inv}(O)$ la date d'émission d'un appel d'une opération O et soit $t_{res}(O)$ la date de réception de la réponse. Alors

$O \rightarrow O'$ si $t_{res}(O) < t_{inv}(O')$ et $O \parallel O'$ si $\neg (O \rightarrow O')$ et $\neg (O' \rightarrow O)$

Linéarisabilité

La cohérence intrinsèque d'un objet x est définie vis-à-vis d'une suite d'opérations sur x. Par exemple :

x est une pile. Conditions de cohérence :

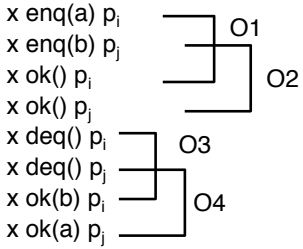
- Si deux objets a et b sont empilés dans l'ordre a ; b, ils doivent être dépilés dans l'ordre b ; a
- Soit ndep(x) le nombre total d'opérations "dépiler" et nemp(x) celui des opérations "empiler" ; alors nemp(x) - ndep(x) ≥ 0. Si la pile a une taille finie n, alors nemp(x) - ndep(x) ≤ n.

On dit qu'un ensemble d'opérations E sur un ensemble d'objets est **linéarisable** si on peut ranger ces opérations dans une séquence S telle que

- pour tout couple d'opérations O, O' telles que O → O', O précède O' dans S
- la séquence S est légale pour tout objet, i.e. respecte les conditions de cohérence intrinsèque pour cet objet

Linéarisabilité - exemple (1)

Soit un objet x qui est une file FIFO (opérations enq et deq) et soit la séquence d'appels et réponses suivants sur x depuis ou vers p_i et p_j



O1 ll O2, O3 ll O4

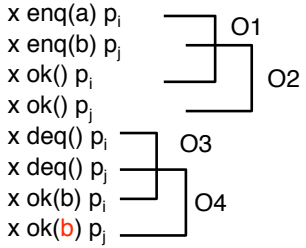
(O1, O2) → (O3, O4)

La séquence est linéarisable en S = [O2, O1, O3, O4]. En effet :

- les conditions de précédence sont remplies
- les conditions de cohérence également : enq(b), enq(a), deq(b), deq(a)

Linéarisabilité - exemple (2)

Soit le même objet x (file FIFO) que dans l'exemple précédent :



O1 ll O2, O3 ll O4

(O1, O2) → (O3, O4)

La séquence n'est pas linéarisable. En effet on ne peut trouver aucune séquence des opérations O1, … O4 qui remplisse la condition de cohérence pour une file FIFO

enq(b), enq(a), deq(b), deq(b)
enq(a), enq(b), deq(b), deq(b)

Linéarisabilité pour des objets dupliqués

Reprenons l'exemple de la file FIFO x avec deux exemplaires et x^2

x^1 reçoit [x enq(b) p_j], [x enq(a) p_i], [x deq() p_i], [x deq() p_j]

x^2 reçoit [x enq(a) p_i], [x enq(b) p_j], [x deq() p_i], [x deq() p_j]

Les deux objets exécutent les requêtes dans l'ordre des réceptions.

Si p_i et p_j prennent les réponse de x^1, l'exécution est linéarisable

Si p_i prend les réponses de x^1 et p_j celles de x^2, l'exécution n'est pas linéarisable

Conditions suffisantes de cohérence

Si une requête est traitée par un serveur, elle doit être traitée par tous les serveurs (en état de marche)

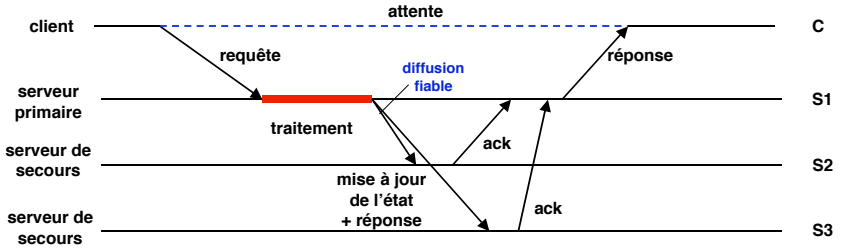
Deux requêtes différentes doivent être traitées dans le même ordre par tous les serveurs

Construction de serveurs tolérants aux fautes

- **Deux techniques de base (nombreuses variantes)**
  - Rappel : N serveurs pour résister à N-1 pannes (franches)
- **Serveur primaire - serveur de secours (poursuite)**
  - Un serveur (primaire) joue un rôle privilégié
  - La panne du primaire est visible par les clients (changement de serveur)
  - La panne d'un serveur de secours est invisible aux clients
- **Redondance active (compensation)**
  - Les N serveurs jouent un rôle symétrique et exécutent tous les mêmes requêtes
  - Les pannes sont invisibles aux client tant qu'il reste un serveur en marche

Les conditions suffisantes de cohérence s'appliquent dans les deux cas

Serveur primaire - serveur de secours

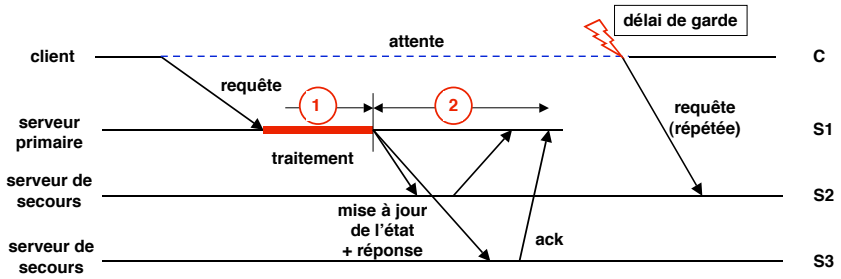


Diffusion fiable : le message parvient à tous ses destinataires non en panne, ou à aucun (pas de diffusion partielle)

Lorsque le primaire renvoie la réponse, tous les serveurs non en panne sont donc dans le même état

Tout serveur de secours peut remplacer le primaire en cas de panne (l'ordre de succession est défini à l'avance, par exemple S2 - S3)

Serveur primaire - serveur de secours
Traitement des défaillances

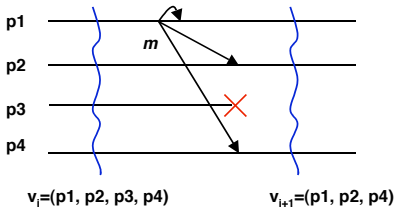


Défaillance du primaire : détectée par le client (délai de garde) et par les serveurs de secours (mécanisme de groupe, défini plus loin)
Le client contacte le serveur de secours qui a pris la place du primaire (il connaît son adresse)
Grâce à la diffusion fiable, tous les serveurs de secours sont à jour, ou aucun.

- Si la panne est arrivée en 1 (aucun serveur de secours n'est à jour) : le nouveau primaire réexécute la requête et met à jour les serveurs de secours
- Si la panne est arrivée en 2 (tous les serveurs de secours sont à jour) : le nouveau primaire renvoie simplement la réponse
- Les requêtes ont une identification unique, ce qui permet de distinguer entre 1 et 2

Serveur primaire - serveur de secours
Mécanisme de base

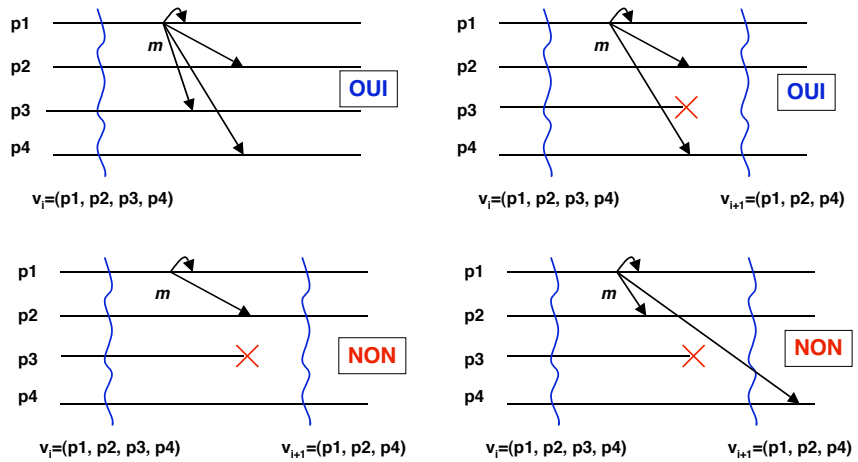
- **Gestion de groupe**
  - Mécanisme permettant de connaître à tout instant la composition d'un groupe de processus (ou serveurs), c-à-d. l'ensemble des processus valides (non en panne)
  - Outil de base : la "vue" : message contenant la liste des processus valides
  - Propriété de cohérence : vues synchrones (cohérence entre diffusion des vues et diffusion des messages)



Soit *m* un message diffusé dans une vue v_i
Soit P = { processus communs à v_i et v_{i+1}}
Alors :

- ou bien *m* parvient à tous les membres de P avant v_{i+1}
- ou bien *m* ne parvient à aucun membre de P

Vues synchrones



Serveur primaire - serveur de secours

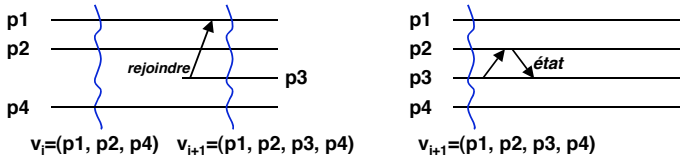
Réinsertion après panne

Après réparation, le serveur défaillant doit se réinsérer et restaurer son état

Réalisé grâce au mécanisme de groupe

1) le serveur qui se réinsère envoie un message *rejoindre* qui provoque un changement de vue

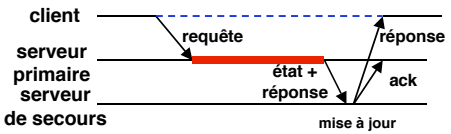
2) le serveur réinséré demande une copie de l'état à un des autres serveurs



Le serveur réinsèré peut fonctionner

Serveur primaire - serveur de secours

Cas particulier de deux serveurs (schéma de Alsberg & Day)



On gagne un peu de temps par rapport au schéma classique (réponse par le serveur de secours)
On n'a pas besoin d'un mécanisme de vues
Le serveur de secours surveille le primaire

En cas de panne du serveur primaire

Détection par le client (délai de garde)
Le serveur de secours devient primaire
Réinsertion du primaire (comme serveur de secours) après réparation, avec copie de l'état

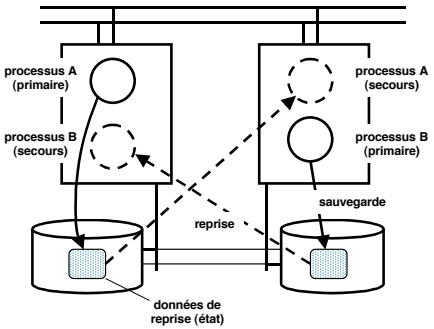
En cas de panne du serveur de secours

Rien à faire par le client
Réinsertion après réparation (copie de l'état)

Le schéma tolère la défaillance d'un serveur

Serveur primaire - serveur de secours

Application à la reprise dans Tandem



Principe de la reprise

- Sauvegarde périodique sur disque de l'état des processus. Surveillance mutuelle de chaque système par l'autre (messages "je suis vivant").
- En cas de dépassement du délai de garde, le processeur survivant reprend les processus défaillants à partir du dernier point de reprise enregistré.

Paires de processus dans Tandem

Échanges entre processus primaire et processus de secours

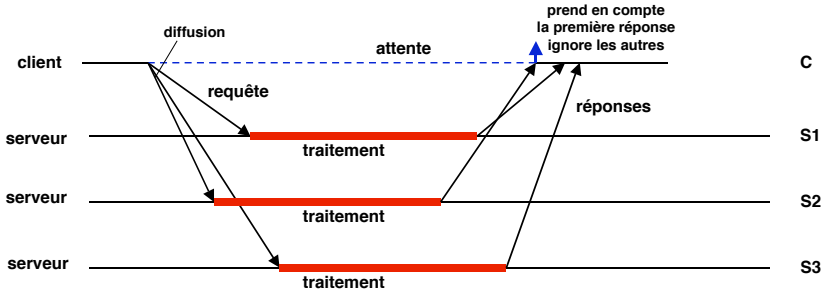
Soit 2 processus primaires A, B ; les processus de secours sont A', B' ; on note [A] l'état de A, etc.

Scénario : requête-réponse de A vers B

Initialement [A] = a0, [A'] = a0, [B] = b0, [B'] = b0
Requête A → B : m (numéroté)
B traite la requête : [B] = b1, [B'] = b0
B met à jour B' : [B] = b1, [B'] = b1
Réponse B → A : (numéroté)
A traite la réponse : [A] = a1, [A'] = a0
A met à jour A' : [A] = a1, [A'] = a1

Une défaillance est tolérée en tout point de cette séquence
La numérotation des messages évite les duplicata

Redondance active

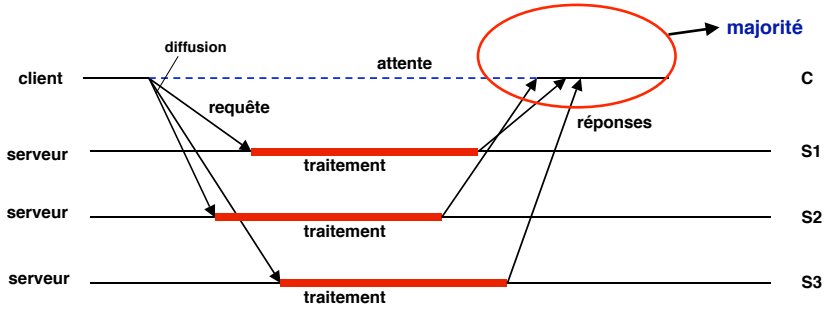


Tous les serveurs sont équivalents et exécutent le même traitement

La diffusion doit avoir les propriétés suivantes (pour satisfaire la cohérence)
- **Un message diffusé est reçu par tous les destinataires (non en panne), ou par aucun**
- **Deux messages différents sont reçus dans le même ordre par leurs destinataires**

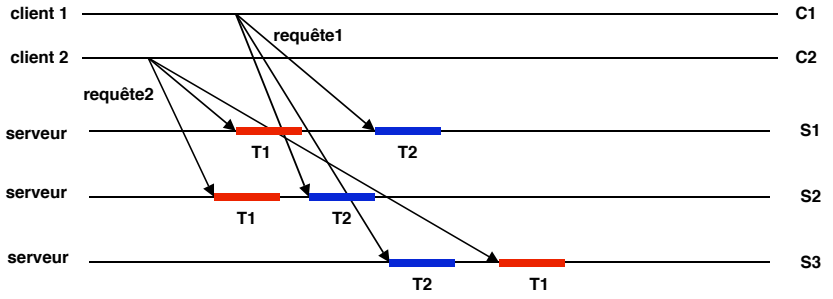
Ces propriétés définissent la diffusion atomique, ou totalement ordonnée

Redondance active (complément)



Le schéma de la redondance active s'étend aux pannes byzantines (réponse arbitraire).
Dans ce cas, on attend toutes les réponses et on choisit la réponse majoritaire. Il faut alors 2k + 1 serveurs pour résister à k défaillances (au lieu de k + 1 avec pannes franches)

Redondance active
Nécessité de la diffusion atomique



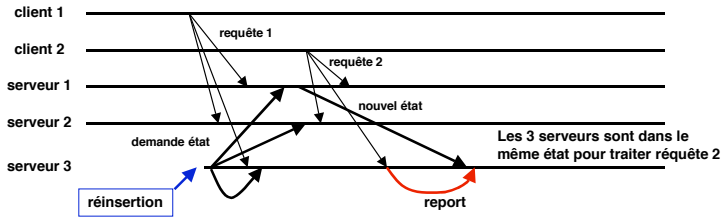
Dans l'exemple ci-dessus, la diffusion n'est pas atomique
Serveur 1 : T1 ; T2
Serveur 2 : T1 ; T2
Serveur 3 : T2 ; T1

Si les traitements modifient l'état des serveurs, le système est incohérent

Redondance active
Réinsertion après panne

La panne d'un serveur est invisible aux clients

Pour se réinsérer après une panne, un serveur diffuse (atomiquement) une demande de reconstitution d'état



Comportement du nouveau serveur
- Un message tel que requête 1 (parvenant avant sa propre demande d'état) est ignoré
- Le traitement d'un message tel que requête 2 (parvenant avant la réponse à la demande d'état) est retardé jusqu'après l'arrivée du nouvel état

Motivation : préserver la cohérence, grâce aux propriétés de la diffusion atomique

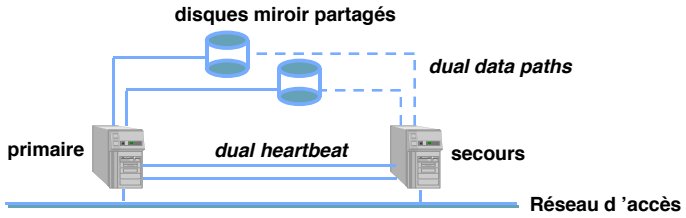
Comparaison entre serveur primaire et redondance active

- **Mécanismes nécessaires**
  - Serveur primaire : technique de groupe dynamique (vue synchrones, sauf si 1 seul serveur de secours, cas très fréquent)
  - Redondance active : diffusion atomique
  - Les deux mécanismes sont équivalents (en complexité)
  - Voir étude détaillée dans la suite du cours
- **Usage des ressources**
  - Serveur primaire : les serveurs de secours peuvent exécuter une tâche de fond, non prioritaire ; reprise non immédiate
  - Redondance active : tous les serveurs sont mobilisés, reprise immédiate
- **Travail pour les clients**
  - Serveur primaire : le client doit détecter la panne du primaire
  - Redondance active : le client n'a rien à faire
- **Conclusion**
  - Serveur primaire : le plus utilisé dans les applications courantes
  - Redondance active : applications critiques, temps réel

Serveurs à haute disponibilité : aspects pratiques

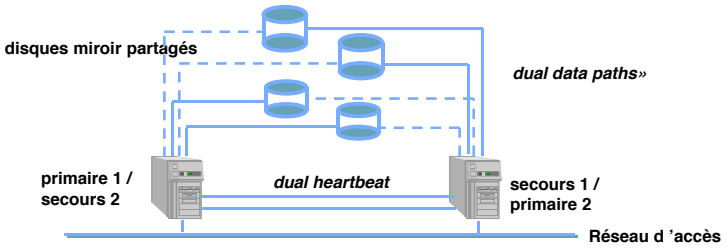
- **Serveur d'information redondant**
  - Duplication passive des serveurs : primaire et secours
  - Réseaux : réseau d'accès, réseau d'administration, réseaux *heartbeat*
  - Disques :
    - Disques privés (par serveur)
    - Disques partagés (redondants -> RAID miroir ou parité)
  - Pas de point de défaillance unique (*single point of failure*)
    - Alimentation électrique
    - Événements externes

Serveurs : configurations (1)



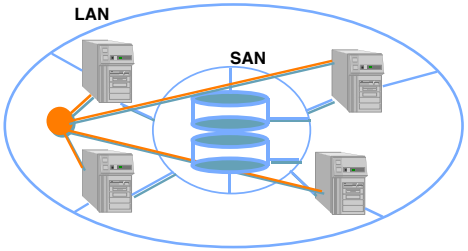
- Configuration 1-1 asymétrique
- Réseaux *heartbeat* dédiés
- Les disques partagés contiennent l'état du système
- Bascule (*failover*) entre primaire et secours en cas de panne du primaire (accès aux disques partagés bascule également)

Serveurs : configurations (2)



- Configuration symétrique
- Les serveurs restent indépendants l'un de l'autre
- Moindre coût
- Impact sur la performance en cas de bascule
- Peut être généralisée à plusieurs applications (*service level failover*)

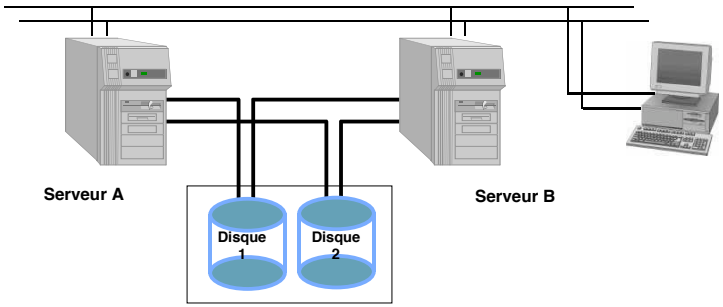
Serveurs : configurations (3)



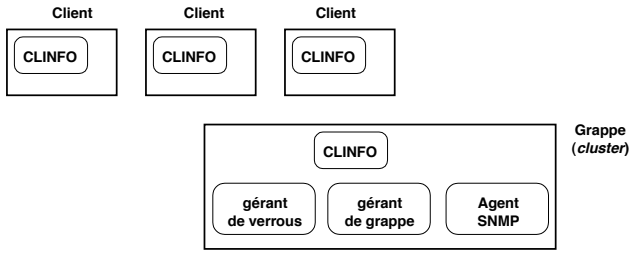
- Accès aux disques miroir via un SAN
- Réseaux *heartbeat* séparés
- Flexibilité accrue
- Utilisation pour serveurs en grappes (*server farms*)

Exemple de système à haute disponibilité : IBM HA-CMP

- HA-CMP : *High Availability Cluster MultiProcessing*
  - Principe : disque partagé (accès multiple)



IBM HA-CMP : architecture logicielle (1)



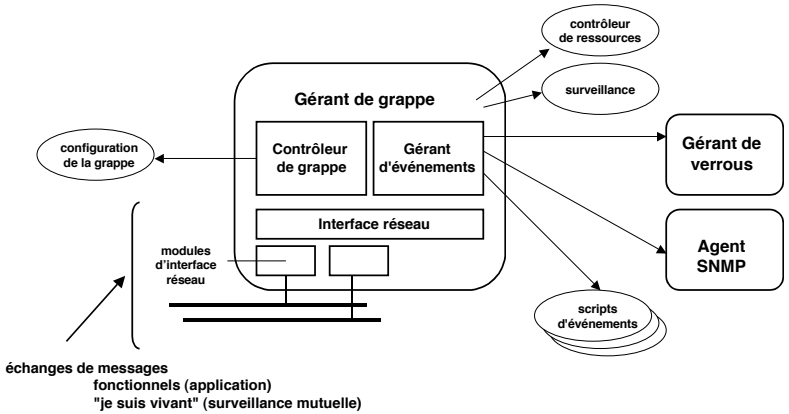
CLINFO : service d'information sur la grappe (informe les clients sur l'état de la grappe)

Agent SNMP : reçoit information du gérant de grappe et la rend accessible via SNMP (*Simple Network Management Protocol*, protocole d'administration des systèmes Unix)

Gérant de verrous : synchronisation distribuée pour la gestion des verrous des transactions

Gérant de grappe : gestion globale de ressources (voir suite)

IBM HA-CMP : architecture logicielle (2)



IBM HA-CMP : principe de la reprise

Plusieurs variantes

- **Recouvrement simple : l'application fonctionne sur l'un des systèmes (primaire). L'autre système est inactif ou exécute des tâches de fond, non critiques.**
- **Recouvrement mutuel : les applications sont exécutées sur les deux systèmes, mais sans partage d'information (fichiers disjoints). En cas de défaillance d'un système, ses applications sont reprises par l'autre.**
- **Partage de charge : les applications sont exécutées sur les deux systèmes et peuvent partager des données. Mais ce partage passe nécessairement par un gérant distribué de verrous pour assurer des propriétés transactionnelles**

Dans tous les cas, la communication (pour reprise) se fait toujours par le disque partagé.