Algorithmique et systèmes répartis

Simulation répartie

Gérard Padiou

Département Informatique et Mathématiques appliquées
ENSEEIHT

5 décembre 2012

plan

Plan

Simulation répartie

Objectif : Simulation exploitant des architectures réparties

- Plusieurs simulateurs peuvent communiquer, coopérer ;
- Exploiter le parallélisme offert par l'architecture ;
- Difficulté : coût des communications et synchronisation des simulateurs.

Hypothèses générales

- Simulation à événements : chaque événement est daté dans un temps global propre à la simulation ($\Rightarrow$ horloge) ;
- Ordre total sur les événements (simultanéité possible) ;
- Variables d'états globales des entités simulées ;
- Principes proches d'un calcul diffusant :
  - Étape initiale produisant un ou plusieurs événements ;
  - Chaque événement entraîne un traitement ;
  - Chaque traitement peut engendrer de nouveaux événements ;
  - Tout nouvel événement est futur.
- Terminaison : l'horloge de la simulation avance jusqu'à
  - soit plus d'événement à traiter ;
  - soit au bout d'une durée fixée.

Un exemple : simulation de traffic aérien

Exemple

- Variables d'état : les vols ;
- Traitements :
  - Planification périodique des vols (toutes les x minutes) ;
  - Envol et Atterrissage ;
- Evénements significatifs :
  - Date de planification ;
  - Décollage ;
  - Atterrissage.
- Evénement datés dans un temps global ;
- Terminaison : Simulation d'une journée de traffic.

Algorithme de base

Structures de donnée

- Les variables d'états ;
- Une file ordonnée des événements à traiter $E = \{e_1, e_2, \ldots\}$
- Ordre total des événements $\equiv$ Datation
$$\forall e_i, e_j \in E : d(e_i) \leq d(e_j) \vee d(e_j) \leq d(e_i)$$
- La date courante $\equiv$ date du dernier événement traité ;
- Prochain événement à traiter $\equiv$ tête de la file E.

Hypothèse

- Une étape ne produit que des événements futurs
$$\forall e \in E : \forall e' \in step(e) = \{e_1, \ldots, e_p\} :: d(e) < d(e')$$
- Mais : $\forall e, e' \in E\ d(e) < d(e') \not\Rightarrow step(e)\ \textit{before}\ step(e')$

Algorithme de base

```
/* Etapes de traitement */
SortedSet <Event> initial_step() {...}
SortedSet <Event> step(Event e) {...}
process Simulation { /* Processus de simulation */
  Event next ;
  Date debut = new Date(); Date hc = debut;
  Date fin = new Date(debut.getTime()+durée);
  SortedSet <Event> EVL = initial_step() ;
  while (!EVL.isEmpty()) {
    next = EVL.first(); hc=next.getDate() ;
    if (hc.getTime()>fin.getTime()) break;
    EVL.addAll(step(next)); EVL.remove(next);
  }
  /* Enregistrer les résultats */
}
```

Répartition d'une simulation

Coopération de plusieurs simulateurs

Deux contextes possibles :

- par partitionnement des traitements : tous participent aux mêmes tâches mais interagissent ;
  ☞ Plusieurs simulateurs : chacun simule un aéroport ;
- par complémentarité : simulation dédiées qui se complètent.
  ☞ Un simulateur de traffic et un simulateur météo.

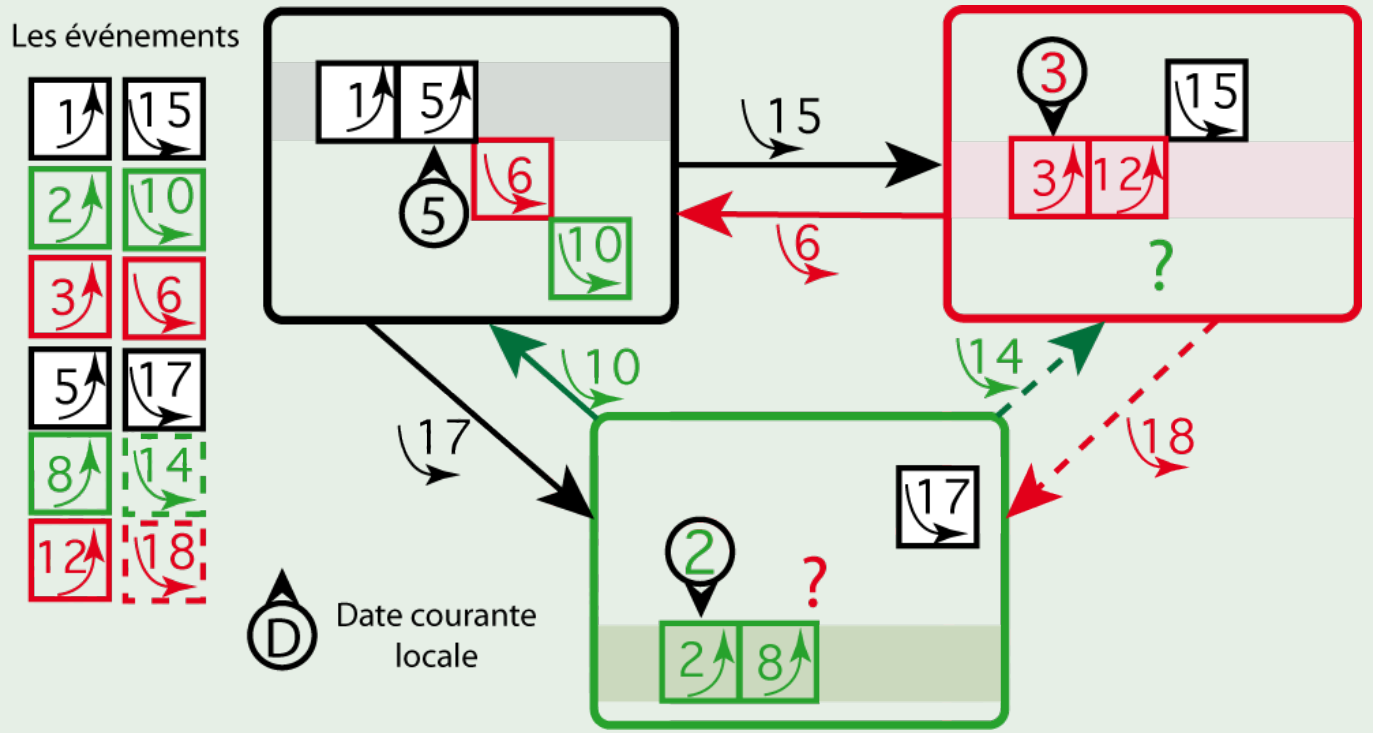
Cohérence globale de la simulation

- Chaque simulateur a une date courante ;
- Comment assurer une synchronisation des simulateurs ?

Le problème de la synchronisation des simulateurs

Une tentative : un état possible . . .

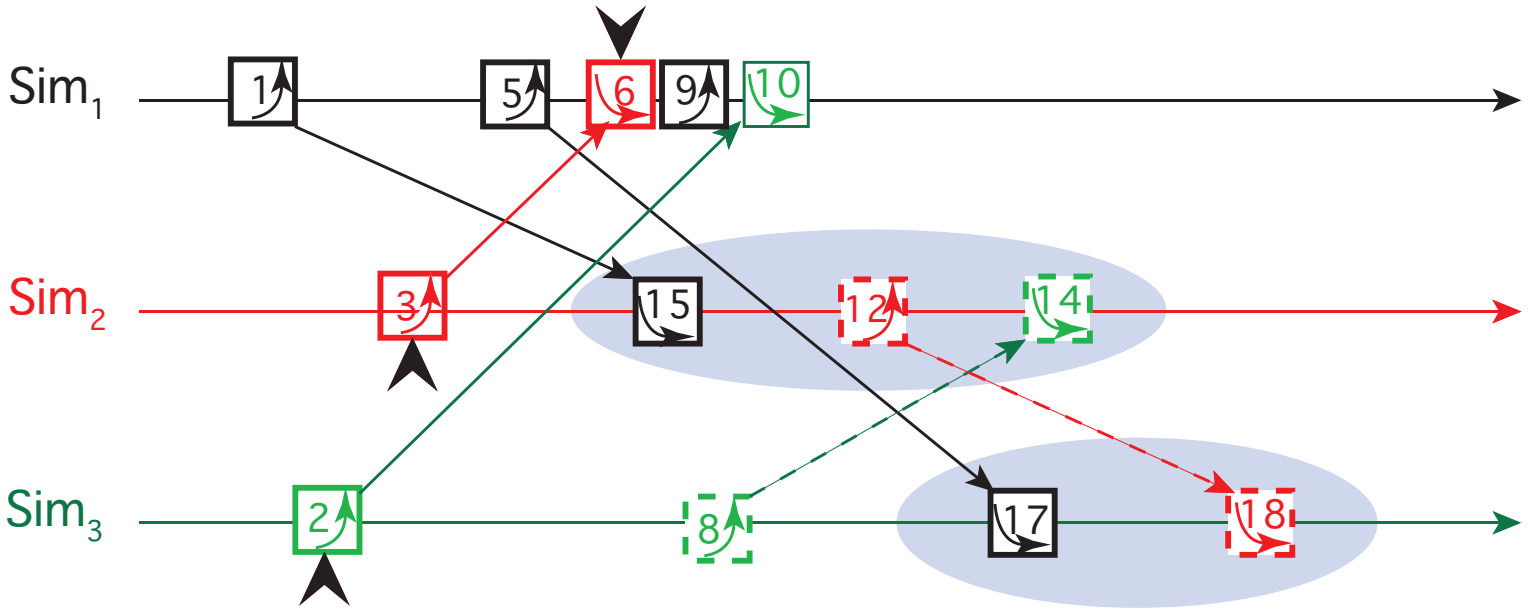
Exemple



Hypothèse : chaque simulation d'aéroport commence par un envol.

Le problème de la synchronisation des simulateurs

Point de vue temporel



- Le simulateur 1 peut avancer de 1 à 6 dès qu'il connaît les événements d'arrivée issus des simulateurs 2 et 3 ;
- Sim_2 ne connaît pas le prochain événement issu de Sim_3 ;
- Sim_3 ne connaît pas prochain événement issu de Sim_2 ;

Critère de synchronisation

Principes

- Chaque simulateur a son horloge (sa date) courante ;
- Chaque simulateur doit savoir où en sont les autres ;
  ☞ répartition de la file des événements
- Objectifs :
  - Éviter les oublis, les retours en arrière ;
  - Éviter l'interblocage.

Deux approches

- Optimiste : laisser faire et revenir en arrière si nécessaire ;
- Pessimiste : éviter les retours en arrière.

Critère de synchronisation

Approche pessimiste

Critère d'avancement

Soit un simulateur s avec une suite ordonnée d'événements non traités locaux ou issus des autres simulateurs :

$$E^s = \{e_1, e_2, e_3, \ldots\} \text{ avec } d(e_1) \leq d(e_2) \leq d(e_3), \ldots$$

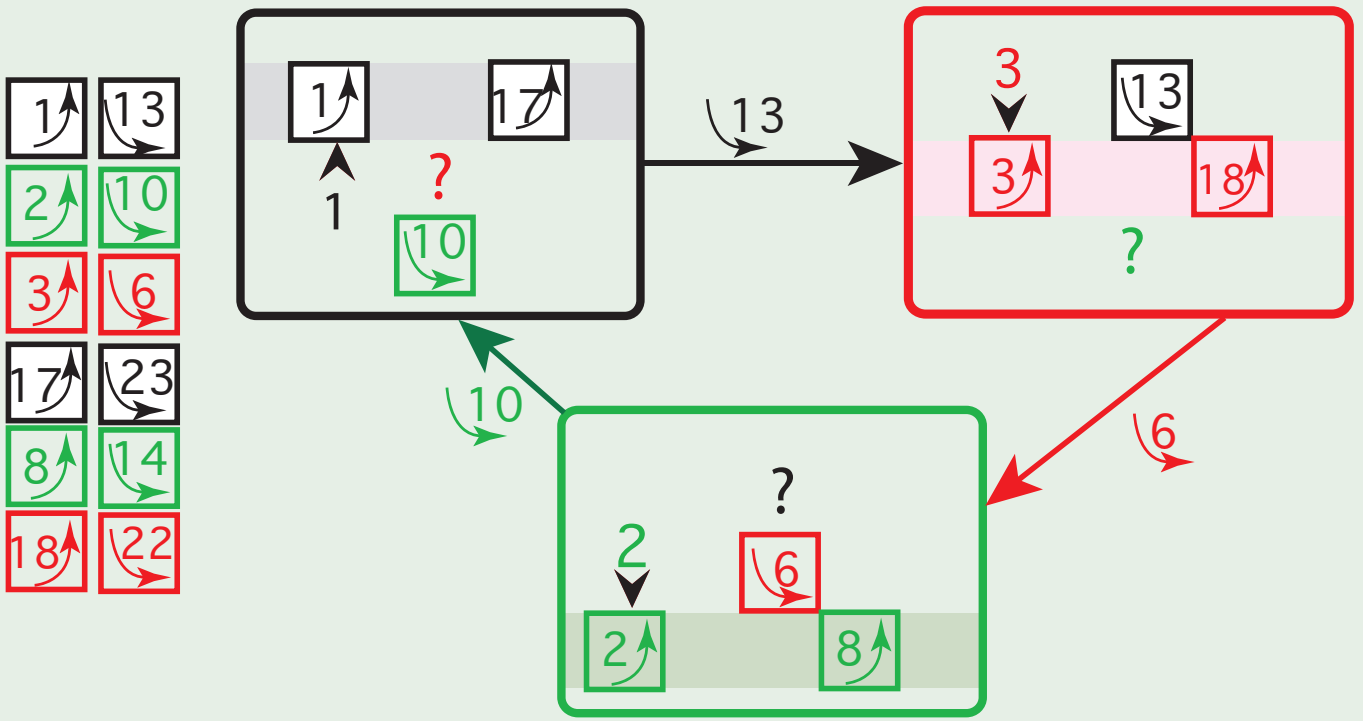
On note $sim(e_i)$ le numéro du simulateur origine de l'événement e_i. L'horloge locale du simulateur s peut être incrémentée pour traiter l'événement suivant e_1 de l'ensemble E^s ssi :

$$\forall s' \in \mathcal{S} : \exists e_i \in E^s : sim(e_i) = s'$$

$\equiv \exists$ dans E^s un événement issu de chaque simulateur participant

Le risque d'interblocage

Exemple

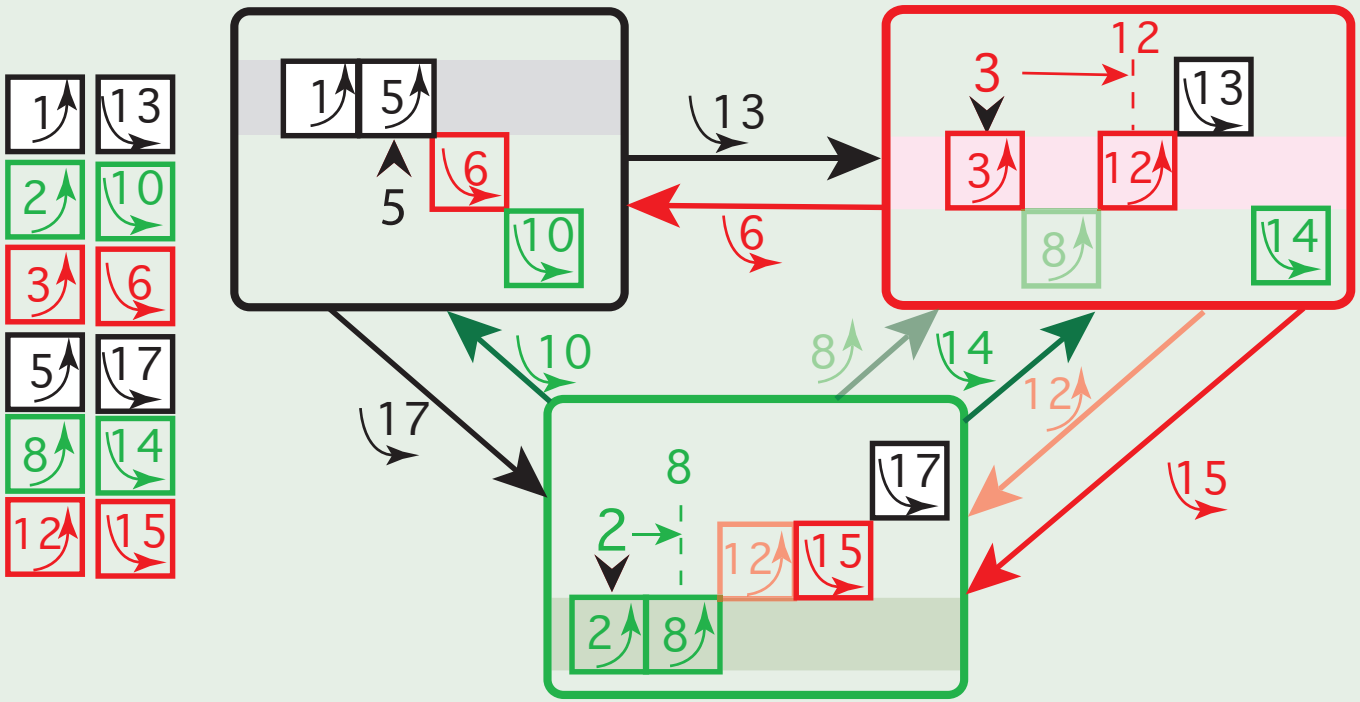


Le risque d'interblocage

L'idée d'anticipation (de lookahead)

Exemple

Les simulateurs rouge et vert s'échangent jusqu'à quand ils ne produiront pas de nouvel événement → débloque le vert



Principe d'un algorithme réparti

Usage de files ordonnées d'événements issus de chaque simulateur

```
process Simulation[s : 1..N] {
 /* Files ordonnées d'évts issus de chaque simulateur */
 SortedSet<Event> E[1..N] = new SortedSet<Event>[N] ;
 for (int i=1; i<=N;i++) E[i]=new SortedSet<Event>();
 E[s] = initial_step() ;
 while (true) {
  int ss = attendre_critere(E) ; /* choix de la file */
  next = E[ss].first(); hc=next.getDate() ;
  if (hc.getTime()>fin.getTime()) break;
  for (Event e : step(next)) /* événements futurs */
    if (sim(e)==s) E[s].add(e) else send(e,sim(e)) ;
  E[ss].remove(next);
 } /* while */
}
```

Principe d'un algorithme réparti (suite)

```
int attendre_critere(SortedSet <Event> E[]){
  boolean vide = true ;
  while (vide) { // risque de boucle infinie→deadlock
    vide=false ;
    for (int s=1 ; s <= N; s++))
         { vide = vide || E[s].isEmpty(); }
  }
  Date min = E[N].getDate() ; int smin = N ;
  for (int s=1 ; s < N; s++)) {
    if (E[s].getDate().getTime() < min.getTime())
      { smin= s; min=E[s].getDate(); }
  }
  return smin ;
}
```

Algorithme évitant l'interblocage (Chandy-Misra-Bryant)

Introduction de messages vides

- Notion de lookahead (anticipation) : durée du prochain intervalle sans événement local nouveau ;
- Utilisation de messages vides datés par :

$$\text{date courante} + \text{lookahead}$$

- Deux possibilités de propagation
  - Par diffusion de messages vides datés après chaque traitement ;
  - Sur demande explicite d'un simulateur ayant une file vide.

Limitations de l'algorithme

Nécessité de résoudre le problème de l'interblocage

- Coût non négligeable ;
- Possibilité de chaînes de messages vides ;
- N'autorise pas un lookahead nul.

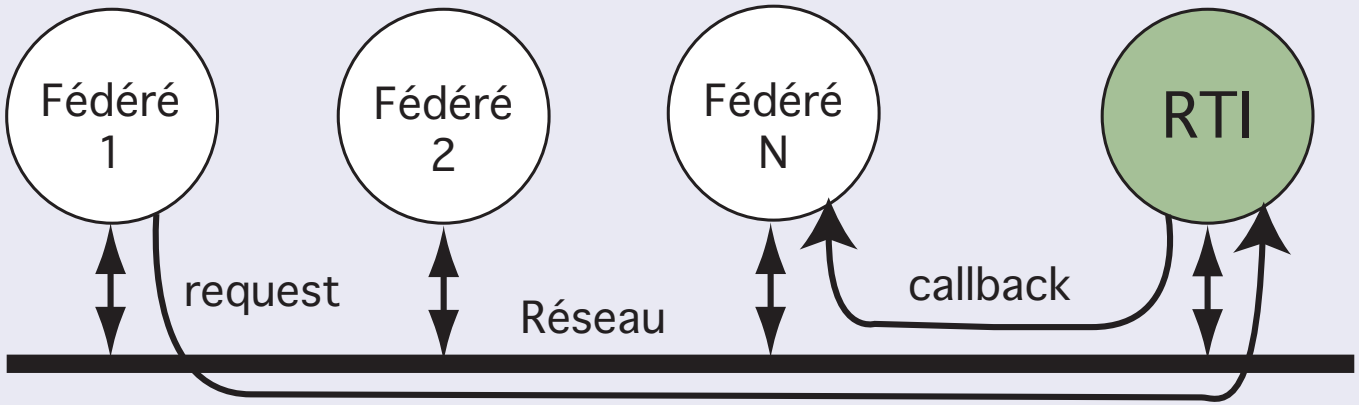
☞ Autre possibilité : détection des interblocages

Plan

La norme High Level Architecture (HLA)

Objectifs

- Spécification d'une API de communication entre simulateurs ;
- Architecture (framework) de référence ;
- Intergiciel support disponible dans différents langages :
  ☞ Runtime Infrastructure (RTI).

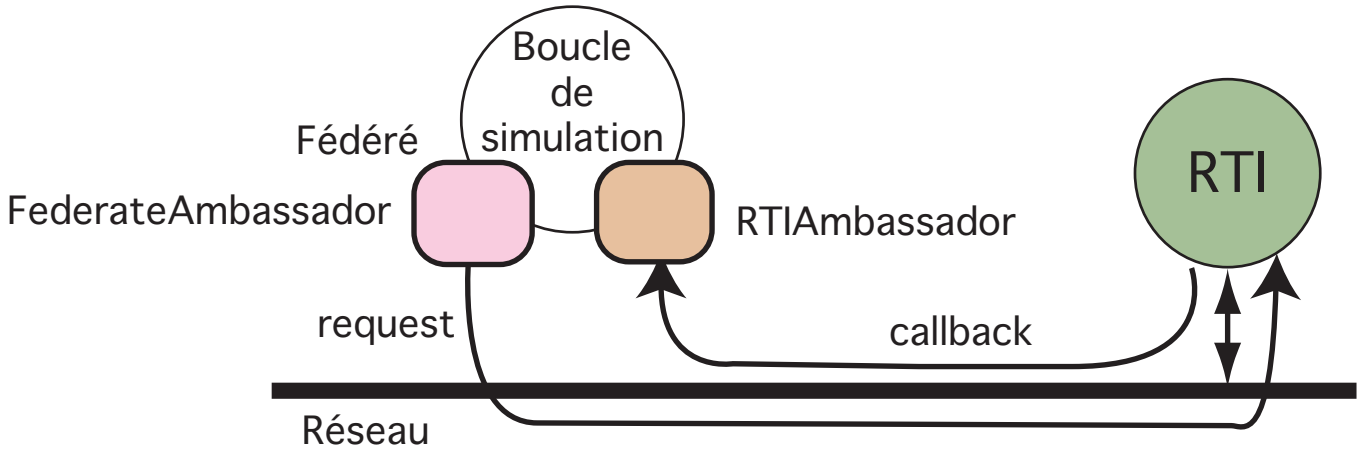


Les concepts de base

- La notion de fédéré $\equiv$ processus lourd de simulation ;
- Fédération : ensemble de fédérés participant à une simulation ;
- Interface entre fédérés définie par des classes et objets exportés ;
- Protocole de communication de type publish/subscribe ;
- Plusieurs politiques de gestion du temps pour ordonnancer les événements.

L'interface de programmation HLA

- La classe `RTIAmbassador` pour les rappels (callbacks) invoqués par le RTI ;
- La classe `FederateAmbassador` pour les services HLA.



Fonctionnalités du RTI

Gestion de la fédération

- createFederationExecution(), destroyFederationExecution() ;
- joinFederationExecution() ;
- registerFederationSynchronizationPoint(), synchronizationPointAcheived().

Gestion des publications/abonnements

- (un)publishObjectClass(), (un)publishInteractionClass() ;
- (un)subscribeObjectClassAttributes(), (un)subscribeInteractionClass().

Fonctionnalités du RTI

Gestion des objets

- registerObjectInstance(), discoverObjectInstance() ;
- deleteObjectInstance(), removeObjectInstance() ;
- updateAttributeValues(), reflectAttributeValues().

Gestion du temps

- enableTimeRegulation(), disableTimeRegulation() ;
- enableTimeConstrained(), disableTimeConstrained() ;
- timeAdvanceRequest(), nextEventRequest().

Les politiques de gestion du temps

- Événements (messages) : mises à jour d'attributs, suppressions, notifications, ...
- Deux types de messages :
  - les messages horodatés : Time Stamped Order (TSO) ;
  - les messages non horodatés : Receive Order (RO).
- Temps local logique propre à chaque fédéré :
  - Un fédéré contrôle l'avancement de son horloge locale ;
  - Aucun fédéré ne doit recevoir de message TSO dans «son passé» ;
  - Le RTI n'autorise une incrémentation d'une horloge d'un fédéré que si celle-ci ne risque pas de mettre en défaut l'invariant précédent.

Les politiques de gestion du temps (suite)

Deux propriétés

- Fédéré contraint : l'avance de son horloge est contrainte par celle d'autres fédérés : il reçoit les messages horodatés TSO ;
- Fédéré régulateur : l'avance de son horloge contraint celle d'autres fédérés : il émet des messages horodatés (TSO).

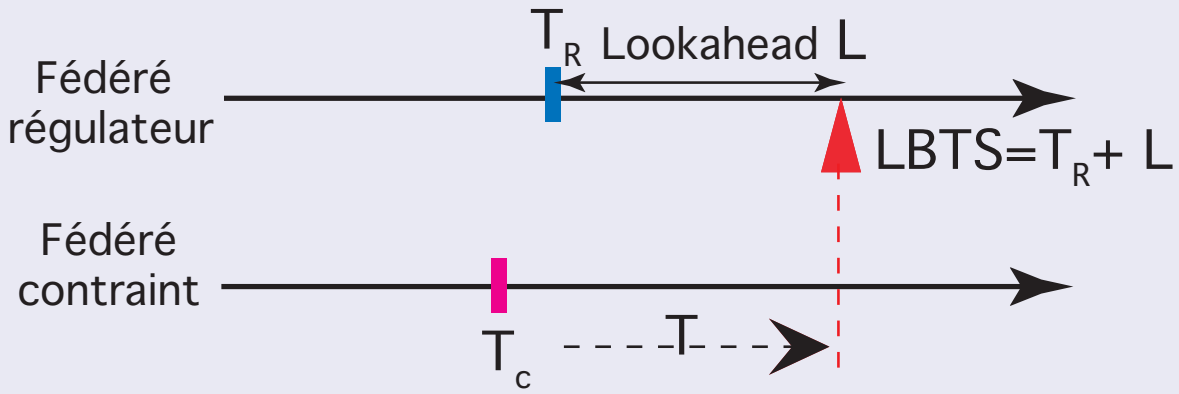
D'où 4 états possibles

- Fédéré régulateur et contraint (émet et reçoit des TSO) ;
- Fédéré régulateur non contraint (émet des TSO) ;
- Fédéré contraint non régulateur (reçoit des TSO) ;
- Fédéré ni contraint ni régulateur (ni l'un, ni l'autre).

Avancement du temps respectant la causalité

Entre un couple (régulateur, contraint)

- T_R : la date courante du fédéré régulateur ;
- T_C : la date courante du fédéré contraint ;
- L : le lookahead déclaré par le fédéré régulateur.



LBTS : Lower Bound on the Time Stamp (aussi appelé GALT (Greatest Available Logical Time)

☞ En général $LBTS = Min(T_{R_i} + L_i)$ de tous les régulateurs F_{R_i}

Plan

Réplication, Temps réel et simulation

Contexte d'étude

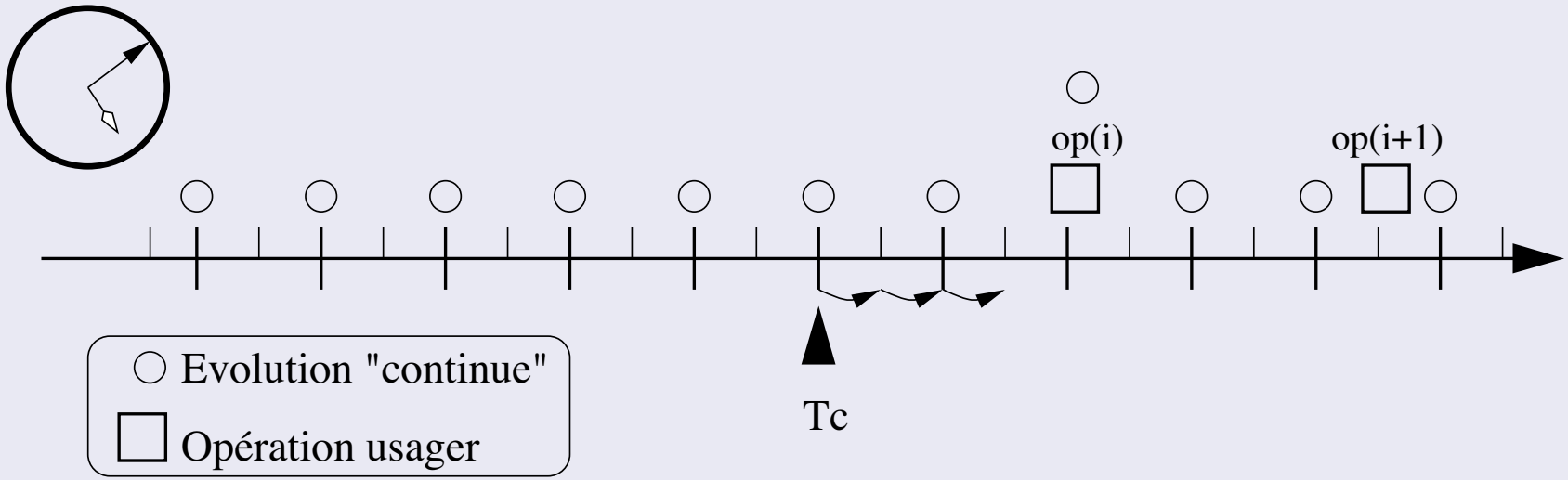
- Domaine : simulation (jeux, réalité virtuelle)
- Réplication : problème de cohérence
- Deux méthodes complémentaires
  - Retard local (Local-lag)
  - Timewarp (« distorsion du temps »)
- Comparaison avec le dead-reckoning (estimation)

d'après : Local Lag and Timewarp : providing consistency for replicated continuous applications, Martin Moore, Jürgen Vogel, Volker Hilt.

Domaine : simulation d'un univers **actif**

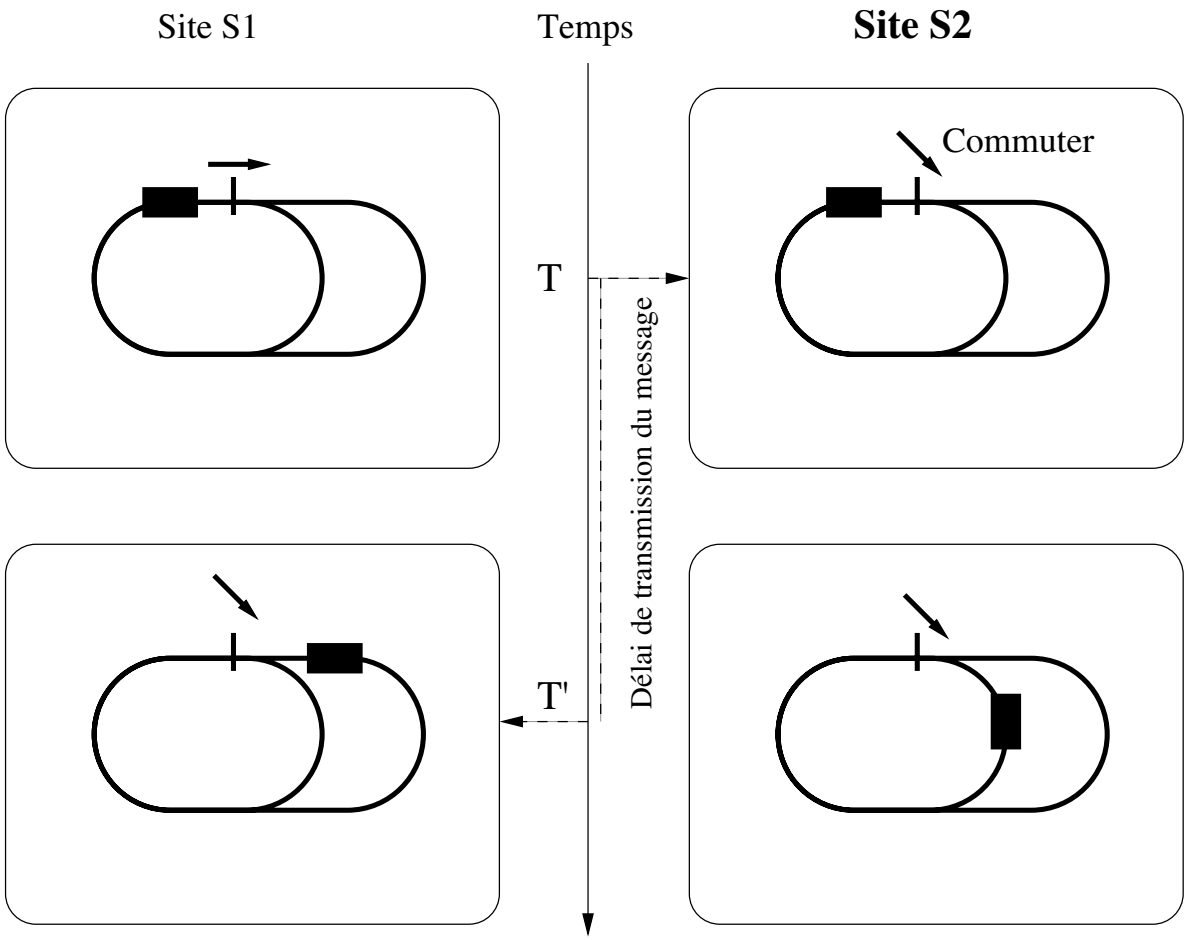
Hypothèses

- Évolution continue dans le temps d'un état global répliqué ;
- État global vu par tous les participants ;
- Chaque opération d'un participant doit être diffusée aux autres ;
- Les opérations sont datées (horloges locales).



Mouvement « continu » et événement local

Idéal (impossible) : Opération connue instantanément par les autres



Technique de maintien de la cohérence

Principes de base

Datation des opérations par l'horloge locale du demandeur et synchronisation des horloges locales (réelles).
Problème Corriger les incohérences dues à l'arrivée trop tardive d'une opération.

- Réagir avec retard : notion de local lag
- Traiter par paquets et corriger si nécessaire : Time-warp
- Calculer le futur probable et corriger si besoin : Dead-reckoning

☞ Point-clé : Compromis risque d'incohérence et temps de réponse

Critère de cohérence de l'état global répliqué

Principe

Toute opération est diffusée à tous les autres sites.

Prédicat $R_i(\ldots)$: « une opération est arrivée à temps » sur i

$$R_i(t, OP(j, T^0, T^*)) = \begin{cases} false & si\ r_i(OP(j, T^0, T^*)) > t \\ true & sinon \end{cases}$$

avec :

T^0 : date de requête et T^* : date d'exécution

$r_i(OP(\ldots))$: date de l'événement de réception de OP sur le site i

Critère de cohérence de l'état global répliqué(suite)

Cohérence globale

Si à l'instant t toutes les opérations sont connues de chaque site, alors l'état de chaque site doit être identique.

$$\forall t, i, j, k : \forall OP(k, T^0, T^*) : T^* < t ::$$
$$R_i(t, OP(k, T^0, T^*)) \wedge R_j(t, OP(k, T^0, T^*)) \Rightarrow E(i, t) \equiv E(j, t)$$

Correction de l'état global répliqué

Principe

Un site idéal virtuel P ordonne totalement toutes les opérations instantanément.

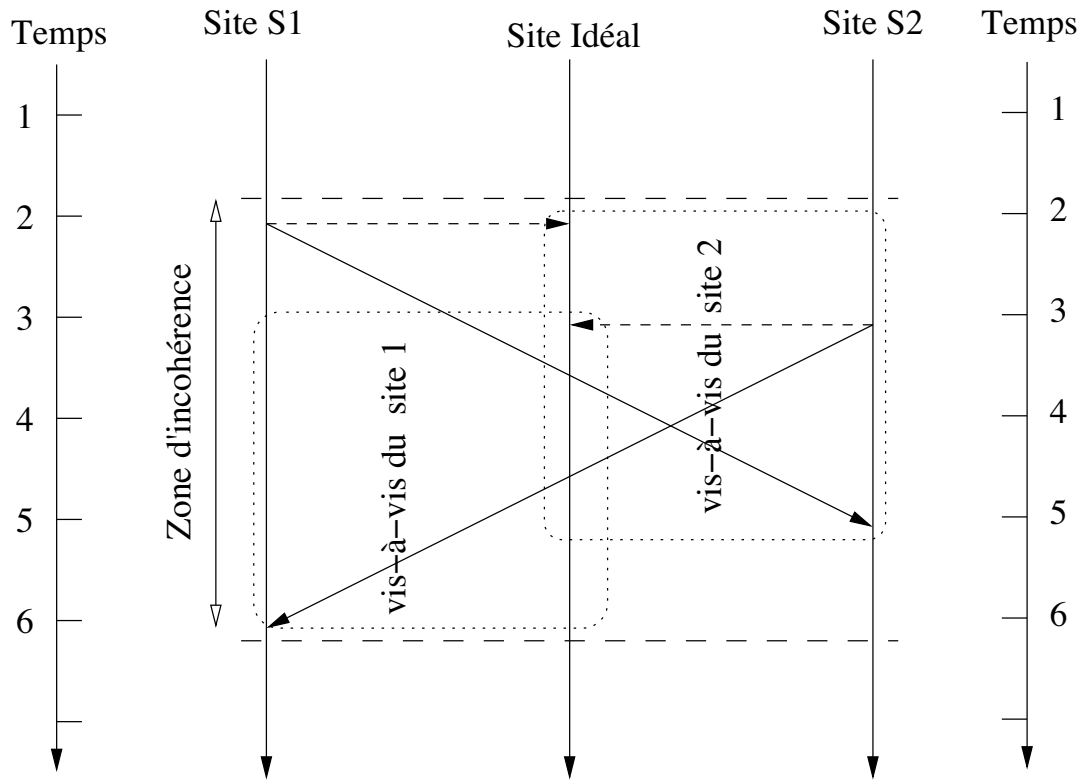
Critère de correction : si à t, toutes les opérations sont connues par le site i, l'état du site i doit être identique à celui du site idéal P.

$$\forall t, i : \forall OP(k, T^0, T^*) \in OPE : T^* < t :: R_i(t, OP(k, T^0, T^*))$$

$$\Rightarrow E(i, t) \equiv E(P, t)$$

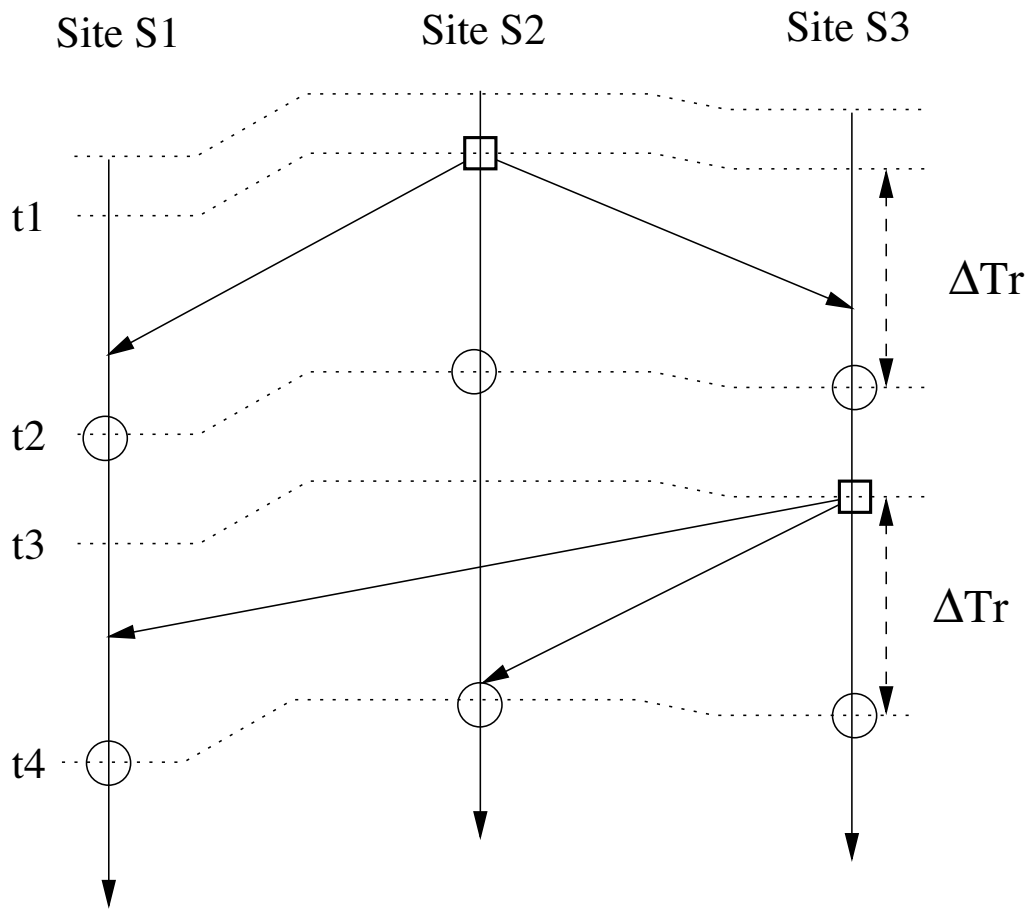
Correction $\Rightarrow$ Cohérence

Cohérence et correction



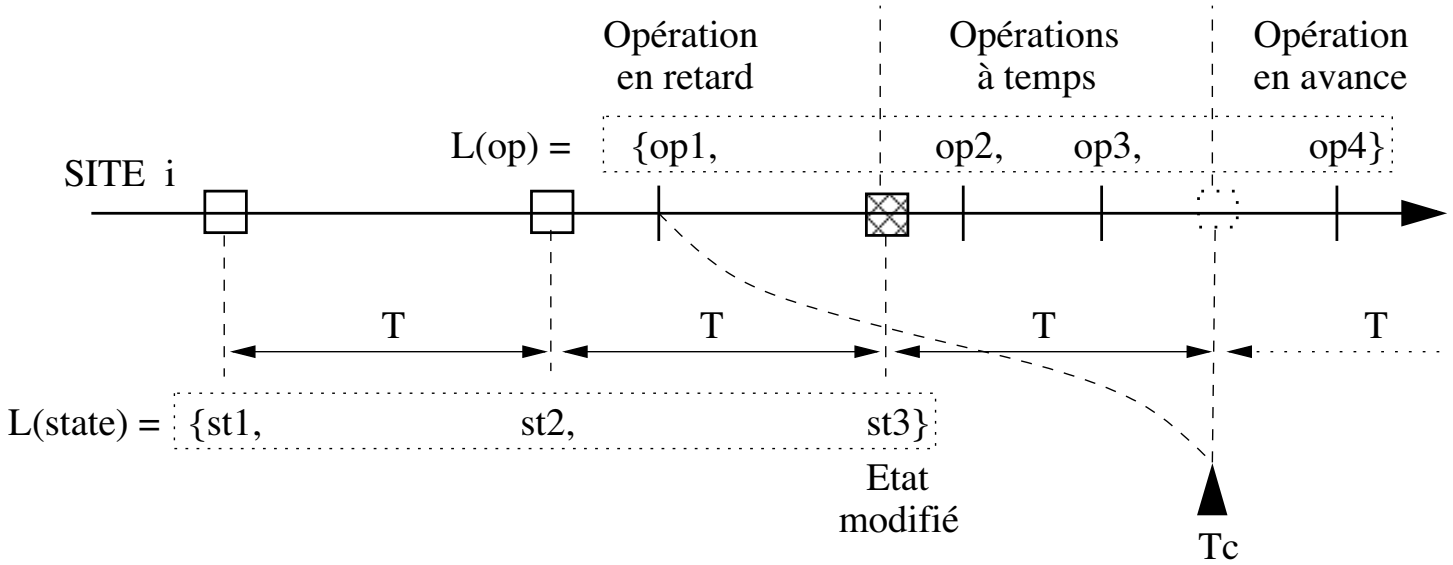
Premiere stratégie : Local-lag

Introduction d'un temps de réponse local minimal



Seconde stratégie : Timewarp

Prise en compte des opérations « en retard »
Algorithme de timewarp



Dead-reckoning

- Transmission des états (au lieu des opérations)
- Objets au comportement prédictible
- Unicité du contrôleur d'un objet
- Diffusion des états imprévus aux abonnés (résultats d'opérations appliquées à l'objet)
- Diffusion périodique (sans action usager) pour palier les pertes de messages.

Remarque Même technique de local-lag applicable à la mise-à-jour de l'état d'un objet.

Conclusion

- Principe de base : retarder l'exécution des opérations
- Corriger "en retard" si nécessaire (problèmes éventuels de réalisme)
- Problème : conserver un temps de réponse acceptable :
  $\mapsto$ réseau à qualité de service « garantie »