

Systemes et algorithmes répartis

Simulation répartie

Gérard Padiou, Philippe Quéinnec

Département Informatique et Mathématiques Appliquées
ENSEEIH

7 octobre 2015



plan

- 1 Simulation à événements
 - Objectif et hypothèses
 - Répartition d'une simulation
 - Synchronisation des simulateurs

- 2 Simulation et temps réel
 - Hypothèses
 - Cohérence des opérations
 - Stratégies



Plan

- 1 Simulation à événements
 - Objectif et hypothèses
 - Répartition d'une simulation
 - Synchronisation des simulateurs

- 2 Simulation et temps réel
 - Hypothèses
 - Cohérence des opérations
 - Stratégies



Simulation

Simulation à événement discret (DES)

- Simulation basée sur des événements
- Exécution au plus vite
- Analyse de systèmes, du futur
- Domaine : réseaux, système de transport, système de fabrication

Simulation d'environnement virtuel

- Environnement interactif
- Simulation temps réel
- Domaine : apprentissage, jeux



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



Quel heure est-il ?

Trois référentiels temporels

- Temps physique : temps dans le système physique que l'on simule
- **Temps de la simulation** : représentation du temps physique dans la simulation (offset par rapport au début de la simulation)
- Temps externe (*wallclock time*) : temps au cours duquel a lieu l'exécution de la simulation

Avancement du temps

- Temps externe : autonome
- Temps de la simulation : par le(s) simulateur(s).
Identique au temps externe (jeu), proportionnel au temps externe, ou le plus vite possible.
- Temps physique : par correspondance avec la simulation

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 trafic aérien

Exemple

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



Algorithme centralisé

Structures de données

- Les variables d'états
- Une file ordonnée des événements à traiter $E = \langle e_1, e_2, \dots \rangle$
- 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
- $step(e) \triangleq$ traitement de e + événements engendrés

Hypothèse

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

Algorithme centralisé

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

Répartition d'une simulation

Coopération de plusieurs simulateurs

Deux contextes possibles :

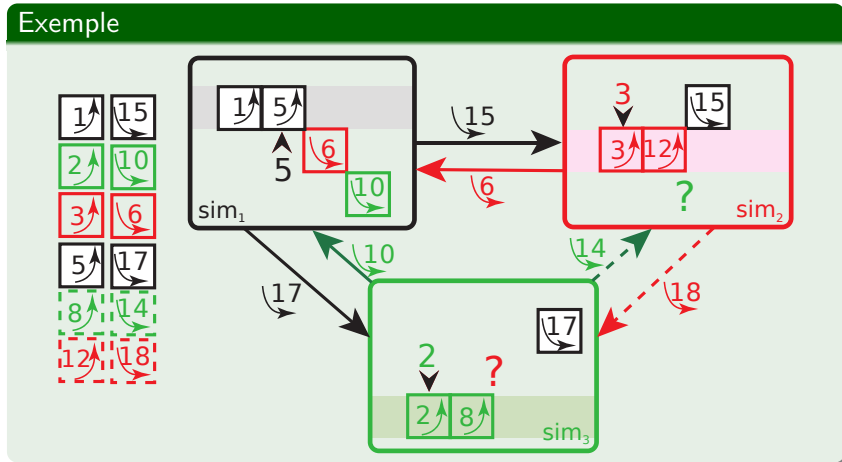
- par partitionnement des traitements : tous participent aux mêmes tâches mais interagissent
⇒ Plusieurs simulateurs : chacun simule un aéroport
- par complémentarité : simulations dédiées qui se complètent
⇒ Un simulateur de trafic et un simulateur météo

Cohérence globale de la simulation

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

Le problème de la synchronisation des simulateurs

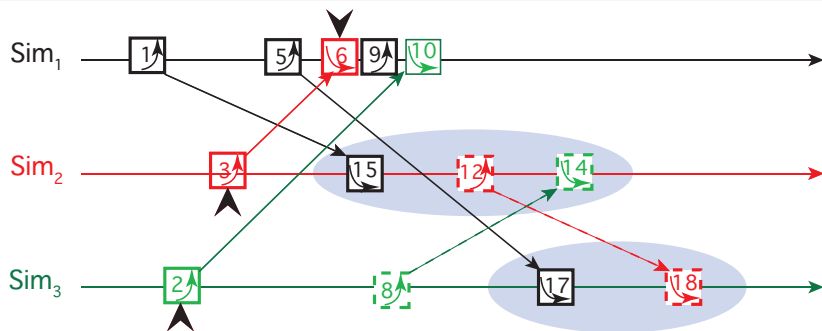
Une tentative : un état possible ...



(hypothèse : chaque simulation d'aéroport commence par un envol) 27

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 le prochain événement issu de *Sim*₂



Critère de synchronisation

Contrainte locale

Sur chaque simulateur, les événements doivent être traités par ordre croissant de leur estampille temporelle

Cette propriété suffit à ce que la simulation distribuée soit équivalente à une simulation centralisée

Principes

- Chaque simulateur a son horloge (sa date) courante
- Chaque simulateur doit savoir où en sont les autres
 - Éviter les oublis, les retours en arrière
 - Éviter l'interblocage

Deux approches

- Pessimiste : garantir le respect de la contrainte, attendre qu'il soit sûr d'avancer
- Optimiste : laisser faire et revenir en arrière si nécessaire

Approche pessimiste

Critère d'avancement

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

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

On note $\text{sim}(e_i)$ le numéro du simulateur **origine** de l'événement e_i . L'horloge locale du simulateur s peut être incrémentée pour traiter l'événement de tête e_1 de la liste E^s ssi :

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

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



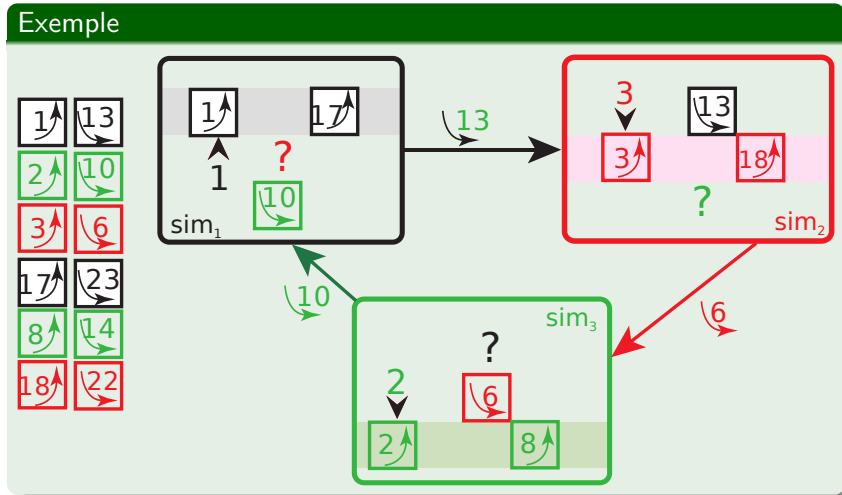
Hypothèse

Hypothèse

- configuration statique, pas de création dynamique de simulateur
- réseau fiable, **FIFO**

⇒ l'estampille du dernier message reçu depuis un simulateur est une borne inférieure (LBTS *lower bound on time stamp*) des messages ultérieurement reçus depuis ce même simulateur

Le risque d'interblocage

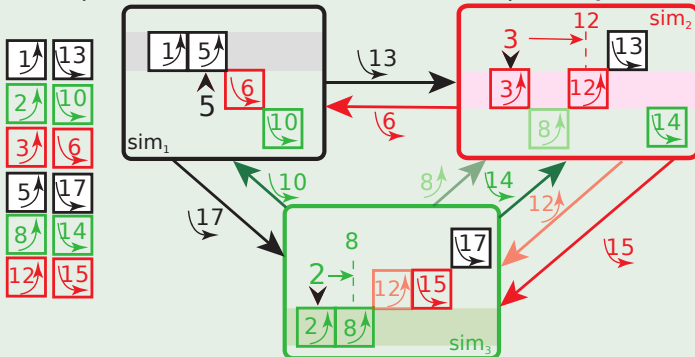


Le risque d'interblocage

L'idée d'anticipation (lookahead)

Exemple

Les simulateurs sim_2 et sim_3 s'échangent jusqu'à quand ils ne produiront pas de nouvel événement → débloquent sim_3



27

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].addAll(initial_step());  
    while (true) {  
        int ss = attendre_critere(E); // choix de la file  
        Event next = E[ss].first(); hc = next.getDate();  
        if (hc.after(fin)) exit;  
        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)

```
// renvoie le n° de la file du prochain évt à traiter
int attendre_critere(SortedSet<Event>[] E) {
    boolean vide = true; // au moins l'un des E[s] est-il vide ?
    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].first().getDate().before(min)) {
            smin = s;
            min = E[s].first().getDate();
        }
    }
    return smin;
}
```

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

Introduction de messages vides

- **Lookahead** (anticipation) : durée du prochain intervalle sans événement **local** nouveau
- Utilisation de messages vides datés par :
date courante + lookahead
- Deux possibilités de propagation
 - Par diffusion de messages vides datés après chaque traitement
 - Sur demande explicite d'un simulateur ayant une file vide



Évitement de l'interblocage par message nul

```
loop {
    int ss = attendre_critere(E);
    Event next = E[ss].first();
    for (Event e : step(next)) // événements futurs
        if (sim(e)==s) E[s].add(e) else send(e,sim(e));
    E[ss].remove(next);

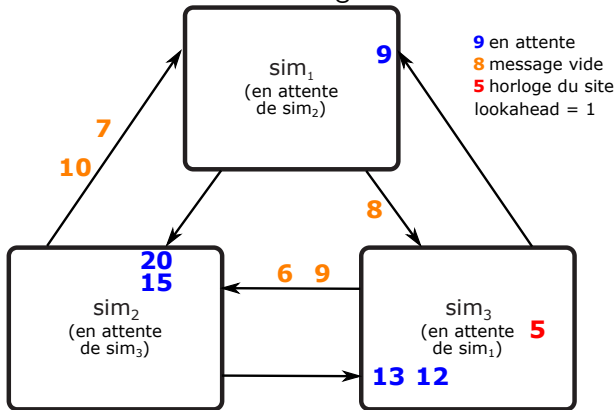
    for (int s=1; s <= N; s++) {
        int lookahead = borne_inf_du_prochain_evt_local;
        int h = current_time + lookahead;
        send(null(h), s);
    }
}
```



Limitations de l'algorithme

Résolution de l'interblocage par message vide

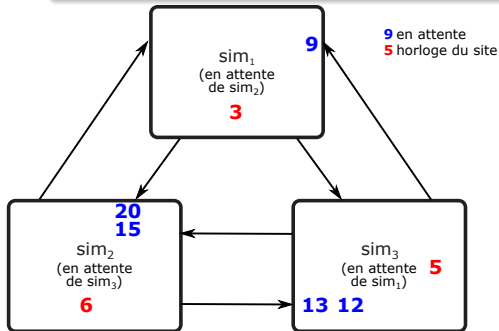
- Estimation du lookahead? (délai minimal entre 2 sites)
- Possibilité de chaînes de messages vides



Amélioration 1 : correction de l'interblocage

Observation

Si tous les simulateurs sont bloqués (interblocage), ils peuvent immédiatement avancer à la date du plus proche événement du système.



Tous peuvent passer à 9.

Solution

- 1 Détection de l'interblocage
- 2 Calcul du min (une vague)
- 3 Diffusion du min

Amélioration 2 : calcul du LBTS

LBTS = lower bound on time stamp = borne inférieure des messages ultérieurement reçus.

Tous les événements de date $<$ LBTS peuvent être traités.

Calcul du LBTS

Pour un **cliché** (snapshot / état global passé), le LBTS est le minimum de :

- des estampilles des messages en transit,
- pour les simulateurs non bloqués, de leur date courante (+ lookahead si connu)
- pour les simulateurs bloqués, de la date de leur prochain événement (+ lookahead si connu)

⇒ calcul asynchrone (en arrière-plan) du LBTS + diffusion du résultat

Plan

- 1 Simulation à événements
 - Objectif et hypothèses
 - Répartition d'une simulation
 - Synchronisation des simulateurs

- 2 Simulation et temps réel
 - Hypothèses
 - Cohérence des opérations
 - Stratégies



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)

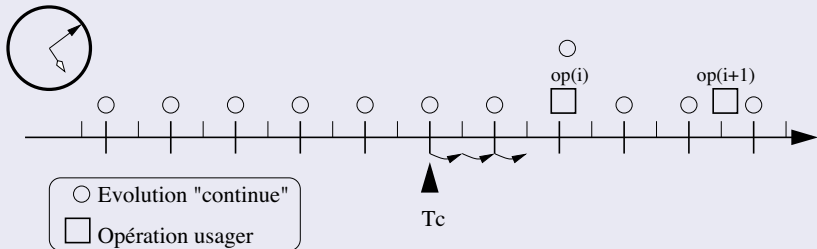
d'après : Local Lag and Timewarp : providing consistency for replicated continuous applications, Martin Moore, Jürgen Vogel, Volker Hilt



Domaine : simulation d'un univers *actif*

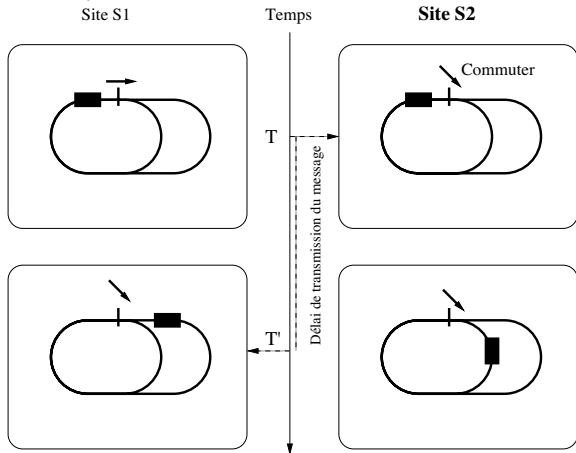
Hypothèses

- 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



Handwritten signature

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 : Timewarp
- 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 $R_i(\dots)$: « une opération est arrivée à temps » sur i

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

avec :

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

$r_i(OP(\dots))$: 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

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

Correction de l'état global répliqué

Principe

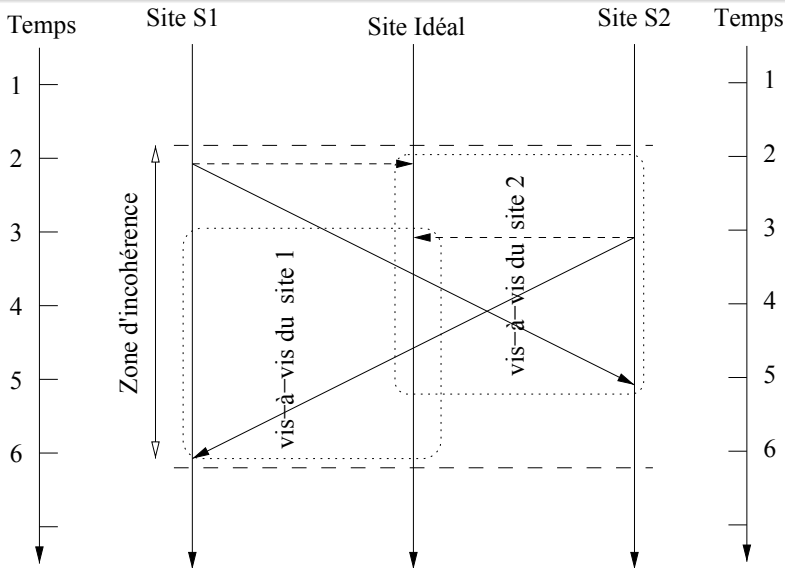
Un site idéal virtuel P ordonne totalement toutes les opérations instantanément

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

$$\forall t, ink : \forall OP(k, T^0, T^*) \in OPE : T^* < t :: R_i(t, OP(k, T^0, T^*)) \\ \Rightarrow E(i, t) \equiv E(P, t)$$

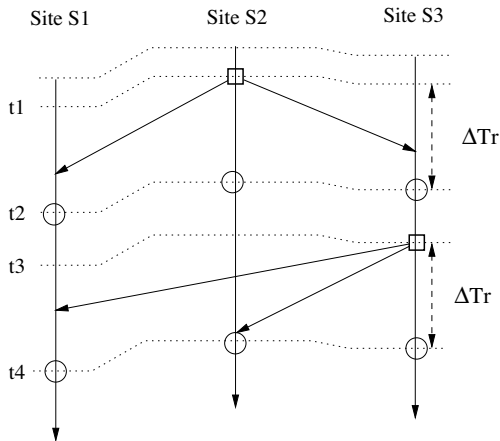
Correction \Rightarrow Cohérence

Cohérence et correction



Première stratégie : Local lag

Introduction d'un temps de réponse local minimal



Un lag > **délat max** de communication garantirait la cohérence



Seconde stratégie : Timewarp

Chaque simulateur fonctionne comme précédemment : il exécute les événements **qu'il connaît** dans l'ordre des estampilles, plus :

- conservation des événements traités,
- mécanisme de sauvegarde de son état (*checkpointing*).

Quand un message en retard arrive (= dans le passé du simulateur), retour en arrière (*rollback*) = annulation des effets des événements traités à tort :

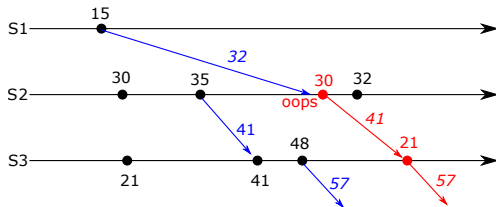
- restauration des variables d'état à la date précédant ce message
- élimination des messages envoyés : **anti-messages**

puis le message en retard est traité.



Timewarp : anti-messages

- Un anti-message correspond à un vrai message (message applicatif) envoyé précédemment
- Quand un anti-message se retrouve dans une file d'attente avec son message correspondant, ils s'annulent
- Quand un anti-message arrive avant son message correspondant, il est mis en attente de ce message
- Quand un anti-message arrive et que son message correspondant a déjà été traité \Rightarrow retour en arrière
- Risque de cascade de retours en arrière (effet domino)



Dead reckoning (navigation à l'estime)

- Transmission des états (au lieu des opérations)
- Objets au comportement prédictible \Rightarrow **estimation** de leur évolution
- 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 – simulation temps réel

- Principe de base : retarder l'exécution des opérations
- Estimer l'évolution en absence d'informations
- Corriger "en retard" si nécessaire \Rightarrow problèmes éventuels de réalisme
- Problème : conserver un temps de réponse acceptable :
 - \rightarrow réseau à qualité de service garantie



Conclusion générale

- Deux types d'applications
 - simulation d'un système pour étudier son comportement
 - simulation d'un environnement virtuel, complémentaire à l'environnement réel
- Système complexe \Rightarrow plusieurs simulateurs interconnectés
- Exemple d'application répartie

