

# Une approche hybride pour le cohérence des données des jeux multijoueurs sur téléphone mobile

Sophie Chabridon



CDUR'05 - Paris

03/11/2005

# Plan de la présentation

1 Introduction .....	3
2 Point sur l'état de l'art .....	7
3 Solution hybride .....	16
4 Conclusion et perspectives.....	23

# 1 Introduction

- Développement des jeux multijoueurs tributaire de la qualité du réseau
- Latence élevée des réseaux de téléphonie mobile 2, 2.5 et 3G
  - ◆ plusieurs secondes
- Latence tolérée par un être humain
  - ◆  $\sim 250$  ms [Pantel and Wolf, 2002]
- Etude des solutions déjà utilisées dans les jeux multijoueurs en réseau fixe
  - ◆ Sont-elles adaptées aux téléphones mobiles (capacité de calcul et mémoire limitées) ?

# 1.1 Caractéristiques des jeux multijoueurs

## ■ Importance de la notion de temps

### ◆ Applications continues

- ▶ Etat modifié par les opérations effectuées ET par le passage du temps

### ◆ Temps de jeu vs temps réel

- ▶ Durée entre deux événements dans le jeu doit rester proche de la durée dans le monde réel
- ▶ Exemple du décollage d'un avion : pas plus de quelques minutes de calcul

## ■ Impact de la latence diffère suivant le type de jeu

- ◆ Exigence la plus forte : Jeux de tir (First-Person Shooter) avec latence tolérée de seulement 150 ms
- ◆ Exigence la plus faible : Au tour par tour

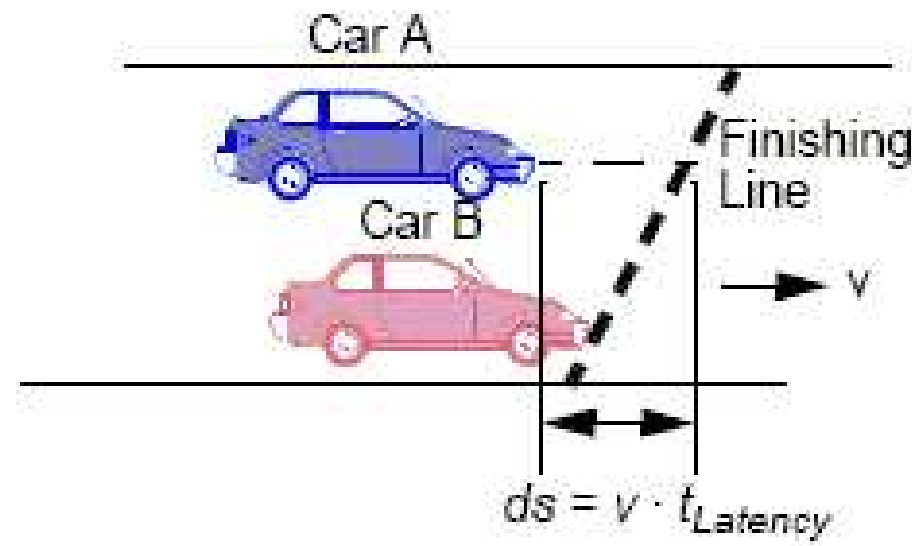
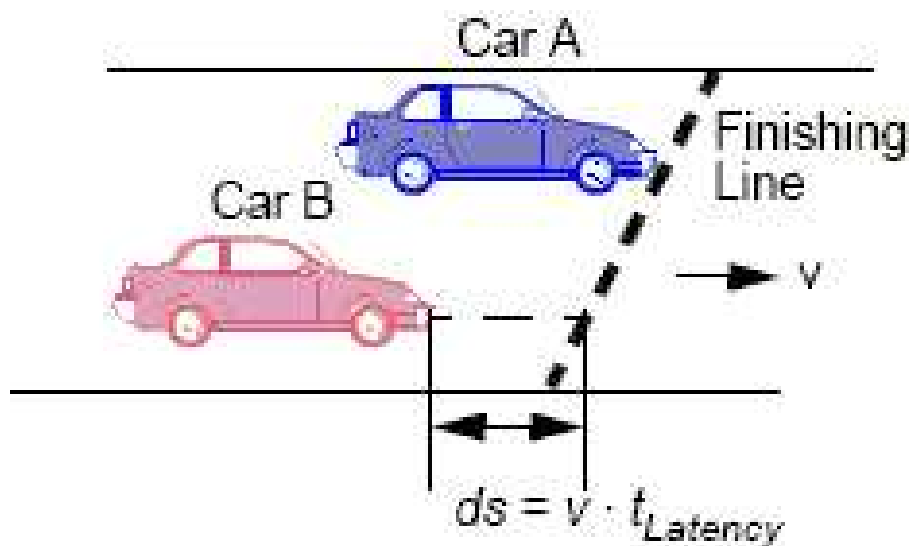
## Caractéristiques des jeux multijoueurs (suite)

- Trois classes de données
  - ◆ Données locales
    - ▶ Propres à un joueur - affichées localement
  - ◆ Données distantes
    - ▶ Besoin de cohérence primordial - pas de copie locale
  - ◆ Données partagées
    - ▶ Besoin de disponibilité primordial - copie locale à synchroniser avec les copies distantes
  
- Priorité à la **jouabilité** au détriment de la cohérence

## 1.2 Impact de la latence - Exemple

■ Cas d'une course de voitures [Pantel and Wolf, 2002]

■ Qui a gagné ?



## 2 Point sur l'état de l'art

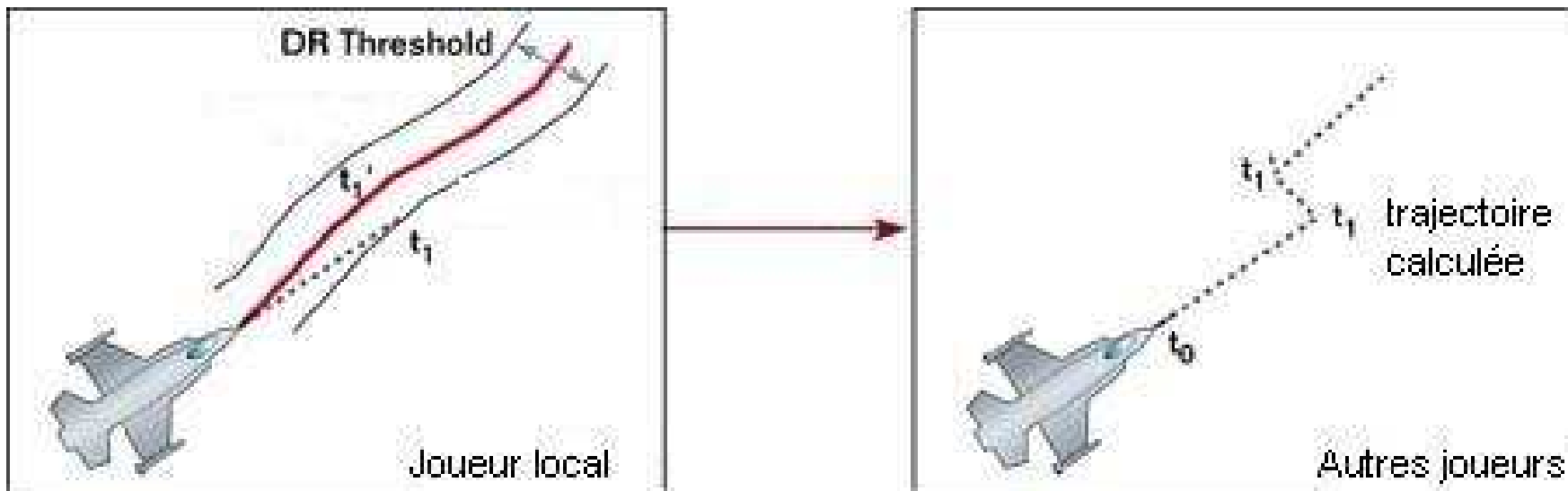
- Solutions en environnement fixe
  - ◆ Méthodes de prédiction d'état
    - ▶ Dead reckoning
    - ▶ Pre-reckoning
  - ◆ Méthodes de synchronisation
    - ▶ Time Warp
    - ▶ Trailing State Synchronisation
- Solution sur téléphone mobile
  - ◆ RendezVous

## 2.1 Dead reckoning

- Signifie "calcul par déduction" (Deduced reckoning)
- Recommandé par la norme IEEE DIS (Simulation Interactive distribuée) [DIS, 1995]
  
- But :
  - ◆ Minimiser le nombre de messages échangés entre les sites
  - ◆ Prédire le mouvement des objets des sites distants sans rafraîchissement systématique
  - ◆ Utiliser uniquement les paramètres connus localement : direction et vitesse de déplacement
  
- Deux étapes :
  - ◆ Un algorithme de **prédiction** du prochain état
  - ◆ Un algorithme de **convergence** après réception d'un message de mise à jour

## Dead reckoning (suite)

### ■ Prédiction du prochain état :



- ◆ Localement, chaque site estime par calcul la position (et/ou l'orientation) des entités distantes impliquées
- ◆ Chaque site gère deux modèles pour une entité locale :
  - ▶ Un modèle représentant son mouvement réel
  - ▶ Un modèle fantôme représentant le modèle prédictif de la position de l'entité sur les sites distants

## 2.2 Pre-reckoning

- Extension du dead reckoning
  - ◆ Anticiper : ne pas attendre que le seuil d'erreur soit dépassé pour envoyer une mise à jour
    - ▶ Probabilité de dépassement du seuil entre le modèle réel et le modèle prédictif
  
- Conçu pour les chemins à grande variabilité
  
- Comparaison des résultats avec le dead reckoning dans un jeu en 3D (CUBE)  
[Duncan and Gračanin, 2003]
  - ◆ Moins de mises à jour
  - ◆ Virages moins marqués
  - ◆ Trajectoire plus précise
  - ◆ Taux d'erreur plus faible

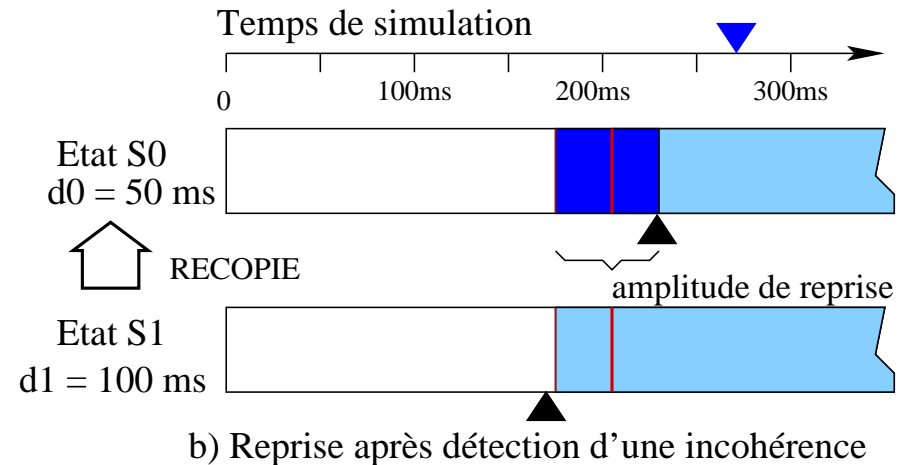
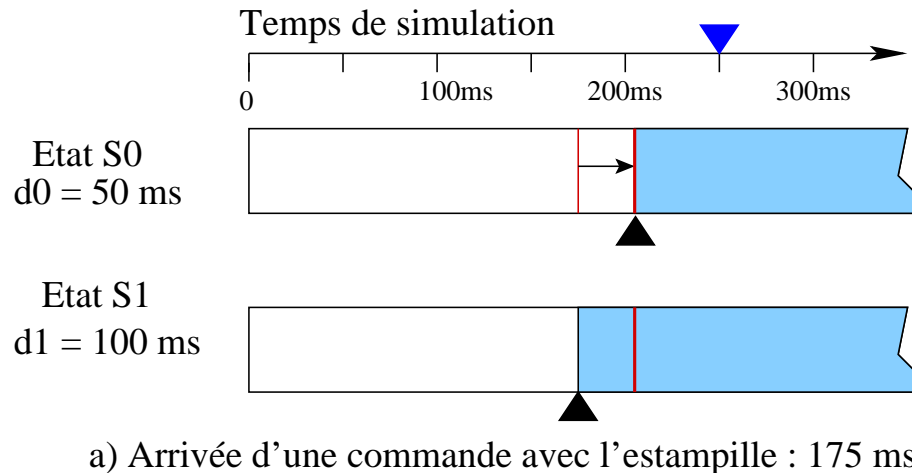
## 2.3 TimeWarp

- Algorithme optimiste prévu pour les simulations militaires interactives
- Principe :
  - ◆ Un **cliché** (snapshot) de l'état de chaque joueur est pris à chaque réception de message
  - ◆ **Retour en arrière** (rollback) si un événement antérieur aux derniers événements exécutés est reçu
  - ◆ Ré-exécution accélérée de tous les événements entre le cliché et l'instant courant
- Optimisations possibles [Mauve et al., 2004]
  - ◆ Clichés périodiques
  - ◆ Retarder la prise en compte des commandes locales pour contre-balancer la latence réseau
  - ◆  $\implies$  Réduit les besoins mémoire, mais rollbacks plus coûteux

## 2.4 Trailing State Synchronization - TSS

- Algorithme optimiste inspiré de TimeWarp
- Créé pour les jeux **FPS** sur des **architectures miroir**
- Chaque site miroir gère plusieurs états de remorquage (trailing states) du même jeu décalés dans le temps
  - ◆ Plusieurs copies du jeu
  - ◆ Exécution de chaque commande mais avec un décalage
- Testé sur le jeu Quake [Cronin et al., 2002]

## Trailing State Synchronization - TSS (suite)



### ■ Détection des incohérences :

- ◆ Après la prise en compte d'une commande, comparaison avec le fil d'exécution précédent (le plus ancien)
- ◆ Incohérence (prise en compte d'une commande à des dates différentes)  $\implies$  retour en arrière (rollback) en recopiant l'état du jeu vers l'état précédent
- ◆ L'état antérieur ré-exécute les événements survenus après l'incohérence jusqu'à l'état courant

## 2.5 RendezVous

- Projet de l'université de Lancaster (UK)
- Plateforme pour les applications distribuées temps réel en réseaux non fiables [Chandler, 2004, Chandler and Finney, 2004, Chandler and Finney, 2005]
- Principe
  - ◆ Réplication optimiste
  - ◆ Pas de rollback
  - ◆ Calcul d'un état cible fictif acceptable par tous les joueurs
  - ◆ Mise en oeuvre d'un arbitre : peut relâcher les règles de jeu pour favoriser la convergence
  
- Développement en C++ sur smartphone
- Tests du mécanisme sur deux jeux mobiles : Pong et Knockabout

## 2.6 Synthèse sur l'état de l'art

Méthode	F/M	Etats fict.	Rollback	Calcul	Mémoire	Cohérence
1. Dead reckoning	Fixe	oui	non	++	-	-
2. Pre-reckoning	Fixe	oui	non	++	-	-
3. Timewarp	Fixe	non	oui	+	++	++
4. TSS	Fixe	non	oui	+	++	++
5. RendezVous	Mobile	oui	non	+	-	+

## 3 Solution hybride

3.1 Mécanisme pessimiste : Délai local .....	17
3.2 Mécanisme optimiste : Contrats prédictifs .....	18

## 3.1 Mécanisme pessimiste : Délai local

- Délai de restitution à l'utilisateur de l'effet d'une action
  - ◆ Dépend de sa perception par le joueur
  - ◆ Délai fonction de la latence réseau pour masquer son existence
  - ◆ Dépend de sa perception par le joueur
    - ▶ Cas d'une course de voitures [Pantel and Wolf, 2002]
      - ★ délai  $< 100ms$  : quasiment imperceptible
      - ★ délai  $> 200ms$  : irréaliste
    - ▶ Délai de restitution de l'effet d'une action locale à conserver assez bas pour ne pas pénaliser la réactivité du joueur
    - ▶ Délai de restitution de l'effet d'une action distante une fois reçue à adapter selon le temps de transfert sur le réseau
  - ◆ Expérimentations en réseau fixe : réduction du nombre d'incohérences [Mauve et al., 2004]
  - ◆ En réseau de téléphonie mobile : forte latence peut attendre plusieurs secondes
    - ▶ Mécanisme insuffisant si utilisé seul
    - ▶ Pénaliserait trop le confort de jeu

## 3.2 Mécanisme optimiste : Contrats prédictifs

- Définis par la norme IEEE DIS (Simulation Interactive Distribuée)  
[DIS, 1995, Mellon and West, 1995]
- Deux méthodes :
  - ◆ 1) Par calcul (*dead-reckoning*)
    - ▶ Basée sur des lois physiques
    - ▶ En l'absence de messages reçus, calcul de la prochaine position à l'aide d'équations (premier ou second ordre)
    - ▶ Utilise uniquement les paramètres connus localement : position courante, direction, vitesse de déplacement...
    - ▶ Mêmes fonctions de prédiction sur chaque site. Limite l'envoi de messages aux cas où un seuil d'erreur est dépassé
  - ◆ 2) Par exécution d'un script
    - ▶ Association de scripts à des événements particuliers sous la forme d'un enchaînement d'actions prédéfinies
    - ▶ Lorsqu'un modèle physique n'existe pas
    - ou
    - ▶ Pour respecter les règles du jeu

## 3.2.1 Exemple

- Animation d'un joueur distant se dirigeant vers un ravin
- Pré-condition sur la position du joueur :
  - ◆  $(pos_x == ravin_x) \& (pos_y == ravin_y)$
- Pour éviter de faire perdre une vie au joueur :
  - ◆ Si la pré-condition est vérifiée
  - ◆ Déclencher une suite d'actions d'attente. Exemple :
    - ▶ *rester sur place*
    - ▶ *se pencher en avant vers le ravin*
    - ▶ *sautiller sur place*

## 3.2.2 Mise en oeuvre

- 1) Cas où un modèle physique existe
- Sur chaque terminal client :
  - ◆ état du jeu vu par le joueur local : 2 modèles
    - ▶ modèle réel des entités locales directement commandé par le joueur
    - ▶ modèle obtenu par calcul (*profil*) pour chaque entité distante
  - ◆ état du jeu vu par les autres joueurs
    - ▶ modèle calculé des entités commandées localement
- Etat du jeu : sous-ensemble des entités partagées entre les joueurs, avec besoin de cohérence et tolérance de divergences transitoires
- Un joueur peut détecter et corriger ses propres divergences au plus tôt
- Si dépassement d'un seuil d'erreur, envoi d'un message de mise à jour aux autres joueurs
- 2) En l'absence d'un modèle physique
- Association d'un ensemble de pré-conditions avec une liste d'actions à réaliser :

$$S = \{precondition \Rightarrow action, touche\_pressee \Rightarrow action, \dots\}$$

### 3.2.3 Détection des divergences

- Définition de valeurs de seuil pour chaque entité de l'état du jeu
- Exemple : position d'un joueur en 2 dimensions
  - ◆ coordonnées réelles :  $pos_x$  et  $pos_y$
  - ◆ coordonnées prédites :  $pred_x$  et  $pred_y$
  - ◆ divergence maximale tolérée :

$$C = \{maxdiv_x, maxdiv_y\}$$

- Test de divergence



$$(|pos_x - pred_x| > maxdiv_x) \parallel (|pos_y - pred_y| > maxdiv_y)$$

- Si dépassement de seuil :
  - ◆ envoi d'un message de mise à jour avec la nouvelle position aux autres joueurs
  - ◆ déclenchement de l'algorithme de convergence par chacun des terminaux distants

## 3.2.4 Algorithme de convergence

- Déclenché par :
  - ◆ réception d'un message de mise à jour
  - ◆ détection d'une divergence entre une entité locale et son profil
- En général, convergence linéaire entre la position prédite et la nouvelle position réelle
- avec respect des règles du jeu
- Exemple de règles :



$$\text{deplacement\_a\_droite} \Rightarrow pos_x = pos_x + 1$$



$$\text{deplacement\_a\_gauche} \Rightarrow pos_x = pos_x - 1$$



$$\text{deplacement\_mur} \Rightarrow rien$$

## 4 Conclusion et perspectives

- Complexité difficile à estimer car liée à l'application
  - ◆ Méthodes de prédiction : Besoin de temps de calcul mais tolèrent de fortes latences voire des déconnexions
  - ◆ Méthodes Timewarp et TSS exigent puissance de calcul et mémoire → pour les terminaux de demain ?
  - ◆ Synchroniser uniquement les données partagées de la zone d'intérêt courante du joueur
- Dépendance très forte avec l'application
  - ◆ Définir des contrats prédictifs généralisables
  - ◆ Identifier les concepts clés ou abstractions
  - ◆ Proposer des patrons de conception
- Grande place pour la créativité : quel comportement est acceptable pour le joueur même s'il ne correspond pas à la réalité ?
- Comment mesurer le confort de jeu ? → Lien avec les études d'usage.

## References

- [DIS, 1995] (1995). Application protocols. In *IEEE Standard for Distributed interactive Simulation*. IEEE Std 1278.1-1995.
- [Chandler, 2004] Chandler, A. (2004). Rendezvous : An alternative approach to conflict resolution for real time multi-user applications. 13th Euromicro Conference on Parallel, Distributed and Network-based Processing.
- [Chandler and Finney, 2004] Chandler, A. and Finney, J. (2004). Rendezvous : The case for a highly optimistic real-time consistency mechanism. In *IEEE WACERTS'04 Proceedings*, Lisbonne, Portugal.
- [Chandler and Finney, 2005] Chandler, A. and Finney, J. (2005). Rendezvous : Supporting real-time collaborative gaming in high latency environments. In *International Conference on Advances in Computer Entertainment Technology ACM SIGCHI ACE 2005*, Valencia and Spain.
- [Cronin et al., 2002] Cronin, E., Filstrup, B., Kurc, A. R., and Jamin, S. (2002). An efficient synchronization mechanism for mirrored game architectures. In *Netgames'02 Proceedings*. Braunschweig, Germany, ACM.
- [Duncan and Gračanin, 2003] Duncan, T. and Gračanin, D. (2003). Pre-reckoning algorithm for distributed virtual environments. In *Proceedings of the Winter Simulation Conference*.
- [Mauve et al., 2004] Mauve, M., Vogel, J., Hilt, V., and Effelsberg, W. (2004). Local-lag and timewarp : Providing consistency for replicated continuous applications. *IEEE Transactions on Multimedia*, 6(1).
- [Mellon and West, 1995] Mellon, L. and West, D. (1995). Architecture optimizations to advanced distributed simulation. In *Proc. of the Winter Simulation Conference*, pages 634–641.
- [Pantel and Wolf, 2002] Pantel, L. and Wolf, L. (2002). On the impact of delay on real-time multiplayer games. In *Proceedings of the 12th Int. Conf. on Network and Operating Systems for Digital Audio and Video*, Miami, Florida, USA.