

**Simulação Distribuída, Time Warp e Variantes:
Descrição e Análise Comparativa**

**Renata Spolon
Roberta Spolon Ulson
Marcos José Santana
Regina Helena Carlucci Santana**

Nº 111

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Abr./2000

SYSNO	<u>1096939</u>
DATA	<u> / / </u>
ICMC - SBAB	

Simulação Distribuída, *Time Warp* e Variantes: Descrição e Análise Comparativa

Renata Spolon *

Departamento de Computação e Estatística - Centro de Ciências Exatas e Tecnologia
Universidade Federal do Mato Grosso do Sul

Roberta Spolon Ulson

Departamento de Computação - Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Marcos José Santana

Regina Helena Carlucci Santana

Departamento de Ciências de Computação e Estatística
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

* Programa de Doutorado - Instituto de Física de São Carlos - Universidade de São Paulo

Abril de 2000

Os autores agradecem as agências Capes, Fapesp e CNPq pelo auxílio ao Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP.

Resumo

A simulação distribuída é uma área de interesse cujo desenvolvimento vem se acentuando nos últimos 20 anos, principalmente devido a fatores como o advento das arquiteturas paralelas e distribuídas, a complexidade dos problemas a serem resolvidos, que passou a exigir tempos inaceitáveis para a execução do programa de simulação e a necessidade crescente por ferramentas de avaliação de desempenho.

Segundo muitos autores, apesar dessa área estar se desenvolvendo nos últimos 20 anos, a comunidade de simulação ainda necessita de ferramentas para possibilitar desde o desenvolvimento do programa de simulação e o particionamento do problema até a análise dos resultados obtidos. Além disso, o problema de paralelizar ou distribuir a simulação exige mecanismos que efetuem a sincronização entre os processos resolvam o problema de causa e efeito. Dezenas de mecanismos têm sido desenvolvidos, utilizando abordagens otimistas ou pessimistas, o que dificulta ainda mais o trabalho do modelador ou usuário.

Para facilitar a compreensão dos mecanismos otimistas existentes, este trabalho apresenta os principais conceitos de simulação distribuída, os mecanismos mais conhecidos e as variantes do mais famoso mecanismo otimista, o *Time Warp*.

Sumário

1. Introdução.....	01
2. Nomenclatura.....	01
3. Simulação Distribuída.....	02
3.1 Mecanismos Conservativos.....	04
3.1.1 O Protocolo <i>CMB</i> (Prevenção do Deadlock utilizando Mensagens Nulas).....	05
3.2 Mecanismos Otimistas.....	07
3.2.1 <i>Time Warp</i>	07
3.2.1.1 Estratégias de Cancelamento.....	10
3.2.1.2 Estratégias para Salvamento de Estados.....	10
4. Limitação do Otimismo no <i>Time Warp</i>	10
4.1 <i>Lock-Step Time-Driven Synchronization</i>	13
4.2 <i>Time Buckets Synchronization</i>	13
4.3 <i>Breathing Time Buckets (BTB)</i>	14
4.4 <i>Breathing Time Warp (BTW)</i>	16
4.5 <i>Local Time Warp (LTW)</i>	18
4.6 <i>Bounded Time Warp</i>	19
4.7 <i>Filter</i>	20
4.8 <i>SFilter (Single-hop Filter)</i>	20
4.9 <i>Moving Time Windows (MTW)</i>	21
4.10 <i>Adaptive Time Warp Concurrency Control Algorithm (ATW)</i>	22
4.11 <i>Probabilistic Distributed Discrete Event Simulation Protocol</i>	22
4.12 <i>Probabilistic Direct Optimism Control (PADOC)</i>	22
4.13 <i>MIMDIX</i>	23
4.14 Algoritmos Adaptativos <i>NPSI</i>	25
4.15 <i>Unified Distributed Simulation (UDS)</i>	25
4.16 <i>WOLF</i>	25
4.17 <i>Window-based Throttling</i>	26
4.18 <i>Penalty-based Throttling</i>	26
4.19 <i>Adaptive Bounded Time Window</i>	26
4.20 Protocolo Probabilístico Baseado em Custos.....	27
4.21 <i>Local Adaptive Protocol (LAP)</i>	27
4.22 <i>Composite ELSA</i>	27
4.23 <i>Conservative-Optimistic Protocol (CO-OP)</i>	28
4.24 <i>Filtered Rollback Algorithm</i>	28
4.25 <i>Tentative Time Warp (TTW)</i>	28
4.26 <i>Adaptive Throttle Schem</i>	28
4.27 <i>ADAPT</i>	29
4.28 <i>Length-based Blocking</i>	29
5. Tabelas Comparativas.....	29
6. Linguagens e Ambientes para Simulação Distribuída.....	32
7. Conclusão.....	35
8. Referências Bibliográficas.....	35

1. Introdução

A simulação distribuída vem se desenvolvendo muito nos últimos anos, porém muito trabalho ainda se faz necessário para que a comunidade passe a utilizá-la como uma ferramenta comum para avaliação de desempenho.

A principal questão relacionada com a simulação distribuída se refere à necessidade de sincronização entre os processos e para isso a literatura descreve basicamente mecanismos otimistas e conservativos. A grande variedade de mecanismos dificulta o trabalho de escolher um que seja mais adequada às necessidades do problema, e a tarefa de efetuar uma análise comparativa entre eles. Como os mecanismos básicos apresentam vantagens e desvantagens, uma série de mecanismos procura resolver um dos problemas relacionado ao mais conhecido mecanismo otimista, o *Time Warp*, através da limitação do excesso de otimismo.

Este trabalho visa organizar as informações relativas aos mecanismos de sincronização otimistas descritos na literatura, especialmente aqueles que se preocupam em limitar o excesso de otimismo do *Time Warp*. Também são descritos os fundamentos da simulação distribuída e o principal mecanismo conservativo, o CMB, para facilitar a compreensão do mecanismo distribuído. Outros problemas abordados são a questão da nomenclatura utilizada em simulação distribuída e a taxonomia dos mecanismos que limitam otimismo no *Time Warp*.

2. Nomenclatura

A simulação paralela e distribuída tem se desenvolvido bastante nos últimos 20 anos, mas ainda não existe um consenso com relação a nomenclatura. Fujimoto [Fujimoto 1993] denomina a execução de um programa de simulação em uma máquina contendo várias unidades centrais de processamento de *concurrent discrete event simulation* (ou simplesmente simulação concorrente). Isso inclui execução em multiprocessadores fortemente acoplados (simulação paralela) e ambientes computacionais distribuídos (simulação distribuída).

Alguns autores utilizam, de forma genérica, a sigla *PDES* (*Parallel Discrete Event Simulation*) [Fujimoto 1990], enquanto outros utilizam a sigla *DDES* (*Distributed Discrete Event Simulation*) [Chiola; Ferscha 1993, Djemame et al. 1996, Vries 1990].

Segundo Ferscha [Ferscha 1995c] quando o sincronismo imposto pelo princípio *SIMD* é explorado para conduzir a simulação com P processadores (sob um controle centralizado), tem-se simulação paralela. Já a estratégia de simulação que utiliza P processadores com sincronização explícita entre os processos é denominada simulação distribuída.

Fujimoto [Fujimoto 1995] destaca simulação paralela (refere-se a *PADS*, lembrando o evento anual *Workshop on Parallel and Distributed Simulation* - onde muitas publicações refletem o estado das pesquisas nessa área) e *Distributed Interactive Simulation* (*DIS*) que teve sua motivação na necessidade dos órgãos militares por meios mais efetivos e econômicos de treinamento de pessoas, com hardware e pessoas distribuídas como se estivessem em situações de combate. O trabalho dessa comunidade de pesquisa está se expandindo para outras aplicações, como treinamento de controladores de tráfego aéreo e entretenimento. Assim, em contraste com os pesquisadores *PADS* que buscam desenvolver tecnologias e ferramentas para possibilitar que programas de simulação sejam desenvolvidos e executados

no menor tempo possível, o principal objetivo do trabalho da comunidade *DIS* tem sido criar ambientes virtuais realistas utilizados por pessoas geograficamente distribuídas.

Mas apesar desses objetivos distintos, existe certa convergência entre as duas áreas de pesquisa, como por exemplo a necessidade de sincronização e gerenciamento do tempo simulado. A tabela 2.1 faz uma comparação entre a pesquisa na área de *PADS* e a pesquisa na área de *DIS* [Fujimoto 1995].

Característica	<i>PADS</i>	<i>DIS</i>
Requisitos de velocidade	o mais rápido possível	Tempo real
Aplicações típicas	circuitos <i>VLSI</i> , telecomunicações	Treinamento militar, jogos, controle de tráfego aéreo, planejamento de emergência (terremotos)
Medida de desempenho	<i>speedup</i>	Realismo, escalabilidade
Modelo de simulação	modelo único	Aliança (junção) de modelos
Distribuição	<i>site</i> único	Geograficamente distribuído
Rede de comunicação	latência arbitrária; <i>LAN</i> ou multiprocessador	Latência limitada; <i>LAN</i> e <i>WAN</i> não confiáveis

Tabela 2.1: Comparação entre *PADS* e *DIS*.

O trabalho de Bagrodia [Bagrodia et al. 1998] ressalta que o termo simulação distribuída foi utilizado inicialmente para descrever a execução de um programa de simulação em vários processadores, levando em consideração uma rede de computadores. Esse termo também vem sendo utilizado para descrever a interconexão de simulações autônomas, como no caso de aplicações militares (*DIS*), embora nesse caso exista diferença com relação aos interesses da simulação (em *DIS* pode haver perda de mensagens, por exemplo [Bagrodia 1998, Pham; Bagrodia 1998]). O autor ainda ressalta que o termo simulação paralela, ou também simulação paralela e distribuída (*Parallel and Distributed Simulation – PADS*) é utilizado para simulações onde a execução do programa é efetuada em vários processadores, independente da plataforma de hardware utilizada (paralela ou distribuída).

Considerando o termo simulação distribuída mais genérico, e pelo fato de não haver um consenso na literatura, esse termo será adotado neste trabalho.

3. Simulação Distribuída

A simulação distribuída refere-se a execução de um único programa de simulação orientado a eventos em uma arquitetura paralela ou distribuída. Este assunto tem sido muito investigado nos últimos anos, e tal interesse se justifica pelo fato de que grandes simulações em engenharia, economia, ciência da computação e aplicações militares consomem grande quantidade de tempo em máquinas sequenciais [Fujimoto 1990].

A razão da dificuldade de se fazer simulação distribuída torna-se evidente quando se analisa a operação de um simulação sequencial. A simulação sequencial geralmente utiliza três estruturas de dados [Fujimoto 1990]:

- Variáveis de estado que descrevem o estado do sistema;

- Uma lista de eventos que contém todos os eventos pendentes que foram escalonados mas ainda não aconteceram;
- Um relógio global para indicar o tempo de simulação.

Cada evento possui uma marca de tempo e determina alguma mudança no estado do sistema sendo simulado. A marca de tempo indica quando essa mudança deve ocorrer. A lista de eventos é ordenada pelo tempo de ocorrência dos eventos, em ordem não decrescente. O programa de simulação repetidamente remove o evento com a menor marca de tempo da lista de eventos e executa as ações necessárias para a ocorrência do mesmo. Ao terminar, o programa de simulação retira o próximo evento da lista, avança o relógio da simulação para o tempo de ocorrência do mesmo e inicia sua execução. Neste paradigma de execução, é crucial que sempre seja selecionado o evento com a menor marca de tempo (E_{min}) como o próximo evento a ser executado. Isso porque se outro evento for selecionado, este pode alterar variáveis de estado utilizadas por E_{min} . Ou seja, estaria-se simulando um sistema no qual o futuro poderia influenciar o passado. Isso é denominado erro de causa e efeito [Fujimoto 1990, Misra 1996].

Considerando a paralelização de um programa de simulação baseado no paradigma acima, verifica-se que a grande oportunidade para paralelismo surge do processamento de eventos concorrentes em diferentes processadores. Entretanto, um mapeamento direto deste paradigma em uma arquitetura distribuída implica em problemas. Basta considerar a execução concorrente de dois eventos, E_1 e E_2 , com marcas de tempo T_1 e T_2 , respectivamente, e $T_1 < T_2$. Se E_1 altera uma variável de estado que é lida por E_2 , então E_1 deve ser executado primeiro para garantir que não irão ocorrer erros de causa e efeito [Fujimoto 1990].

O sistema sendo modelado, referenciado como sistema físico, é visto como um conjunto de processos físicos que interagem durante a simulação. A simulação é construída como um conjunto de processos lógicos $LP_0, LP_1, \dots$, um por processo físico. Toda a interação entre os processos físicos é modelada através de mensagens de eventos contendo uma marca de tempo, enviadas entre os correspondentes processos lógicos por canais de comunicação. Cada processo lógico contém uma parte do estado correspondendo ao processo físico que modela, assim como um relógio local. [Fujimoto 1990].

Pode-se garantir que não ocorrem erros de causa e efeito se e somente se cada processo lógico executa eventos em ordem não decrescente de marca de tempo. Considere como exemplo, dois eventos, E_1 no processo lógico LP_1 (marca de tempo 10) e E_2 no LP_2 (marca de tempo 20), como mostrado na figura 3.1.a. Se E_1 escalona um novo evento E_3 para o processo LP_2 contendo uma marca de tempo menor do que 20, E_3 poderia afetar E_2 , o que implica na necessidade de execução sequencial dos três eventos (figura 3.1.b). Se um LP não tem informação sobre que eventos poderiam ser escalonados por outros eventos, um LP deveria ser forçado a processar o único evento seguro, aquele contendo a menor marca de tempo, resultando em uma execução sequencial. Ou seja, do ponto de vista do sistema físico, a causa sempre deve preceder o efeito, sendo responsabilidade do mecanismo de simulação garantir esse sequenciamento quando o programa de simulação é executado paralelamente [Fujimoto 1990].

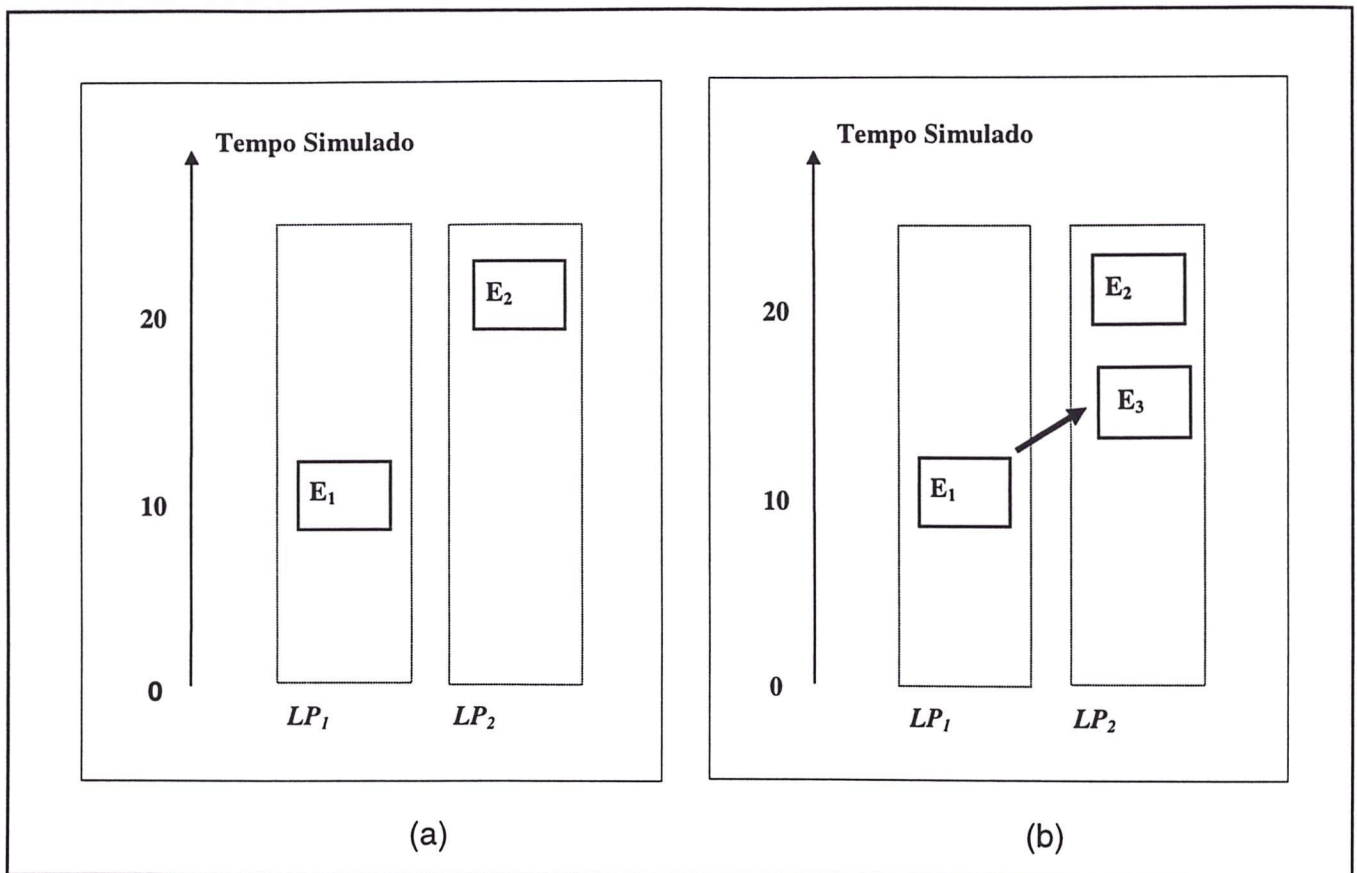


Figura 3.1: Ocorrência de Erros de Causa e Efeito

Para preservar a ordem de causa e efeito entre os eventos, são necessários mecanismos que sincronizem os tempos de simulação dos LPs . Tais mecanismos são classificados em duas categorias: conservativos e otimistas (alguns autores ressaltam uma terceira categoria, a dos mecanismos mistos, com submodelos que executam em modo otimista ou conservativos [Bagrodia 1998]). As abordagens conservativas evitam a possibilidade de ocorrer um erro de causa e efeito, ou seja, estão baseadas em alguma estratégia para determinar quando é seguro processar um evento (isto é, determinar quando todos os eventos que podem afetar o evento em questão tenham sido processados). Os LPs podem processar um evento somente quando as relações de causa e efeito são preservadas. Nesse caso, podem ocorrer *deadlocks* e para evitá-los foram propostos vários algoritmos, explicados a seguir. Já as abordagens otimistas utilizam algoritmos que permitem a simulação progredir o que for possível, e então, através da estratégia de detecção e recuperação, utilizam um mecanismo de *rollback* para recuperar os estados dos LPs se uma violação de causa e efeito ocorrer [Bagrodia 1996, Fujimoto 1990, Hoeger; Jones 1996, Pham; Bagrodia 1998, Radhakrishnan et al. 1998].

3.1 Mecanismos Conservativos

Historicamente, os primeiros mecanismos de simulação distribuída foram baseados em abordagens conservativas. O problema básico que os mecanismos conservativos devem resolver é determinar quando é seguro executar um evento. Ou seja, se um processo contém um evento não executado E_i com marca de tempo T_i (e nenhum outro com marca de tempo menor), e o processo pode determinar que é impossível para ele receber outro evento com marca de tempo menor que T_i , então o processo pode executar E_i seguramente porque pode garantir que fazendo isso não irá resultar em uma violação de causa e efeito. Aqueles

processos que contêm eventos que não são seguros devem ser bloqueados, o que pode levar a situações de *deadlocks* se não forem tomadas precauções adequadas [Fujimoto 1990].

3.1.1 O Protocolo CMB (Prevenção do Deadlock utilizando Mensagens Nulas)

Os primeiros algoritmos para simulação paralela desenvolvidos de maneira independente por Chandy e Misra [Chandy; Misra 1979] e Bryant (por isso a denominação *CMB*) [Ferscha 1995b] exigiam que a especificação dos canais que indicam a comunicação entre os processos (topologia) fosse feita estaticamente, e também que as mensagens chegassem em cada canal em ordem de marca de tempo. Para determinar quando é seguro processar uma mensagem, é exigido que a sequência de marcas de tempo de mensagens enviadas em um canal seja não decrescente. Isso garante que a marca de tempo da última mensagem recebida em um canal de entrada seja um limite inferior da marca de tempo de qualquer mensagem que irá ser recebida posteriormente [Fujimoto 1990].

Cada canal possui um relógio associado que é igual a marca de tempo da mensagem da cabeça da fila, se a fila contém uma mensagem, ou a marca de tempo da última mensagem recebida, se a fila está vazia. O relógio do *LP* (*Local Virtual Time - LVT*) é sempre o menor valor entre os relógios dos seus canais de entrada, ou zero se nenhuma mensagem tiver sido recebida (figura 3.2).

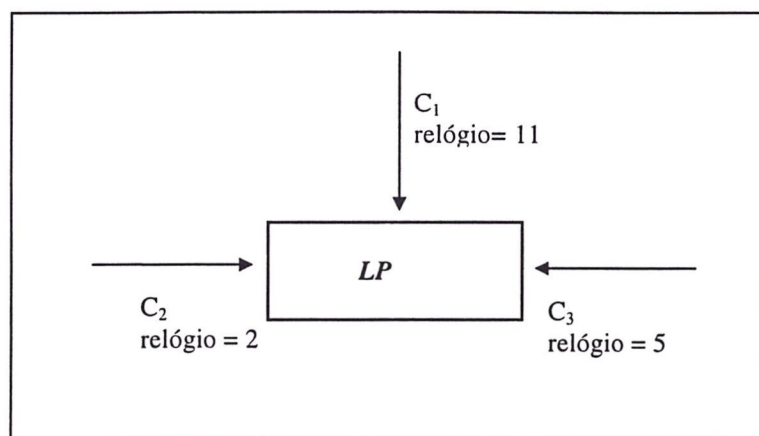


Figura 3.2: *LVT* de um Processo Lógico.

O *LP* seleciona repeditamente o canal com o menor relógio e, se houver mensagem na fila, executa-a. A ordem de processamento dos eventos estará correta porque todas as mensagens futuras recebidas terão marcas de tempo com valores posteriores ao relógio local, já que elas irão chegar em ordem cronológica ao longo de cada canal. Se a fila selecionada está vazia, o *LP* bloqueia. Isso porque o mesmo pode receber uma mensagem deste canal contendo um tempo que é menor que todos as outras marcas de tempo de entrada. Para garantir essa ordem cronológica, o *LP* é forçado a esperar por uma mensagem para atualizar o relógio no canal antes que possa atualizar seu relógio local [Fujimoto 1990, Misra 1986].

Um problema crítico nesse protocolo ocorre quando surge um ciclo de filas vazias com valores de relógios suficientemente pequenos. Cada *LP* no ciclo se bloqueia, esperando indefinidamente por uma mensagem no canal de entrada que contém o menor valor de relógio, e a simulação entra em *deadlock*. Os *deadlocks* ocorrem quando existe um ciclo de

LPs bloqueados e cada um está bloqueado devido a outro processo no ciclo. Por exemplo, considere a rede da figura 3.3 [Fujimoto 1990]. Cada *LP* está esperando no canal de entrada contendo o menor valor de relógio porque a fila correspondente está vazia. Todos os três *LPs* estão bloqueados, embora existam mensagens em outras filas.

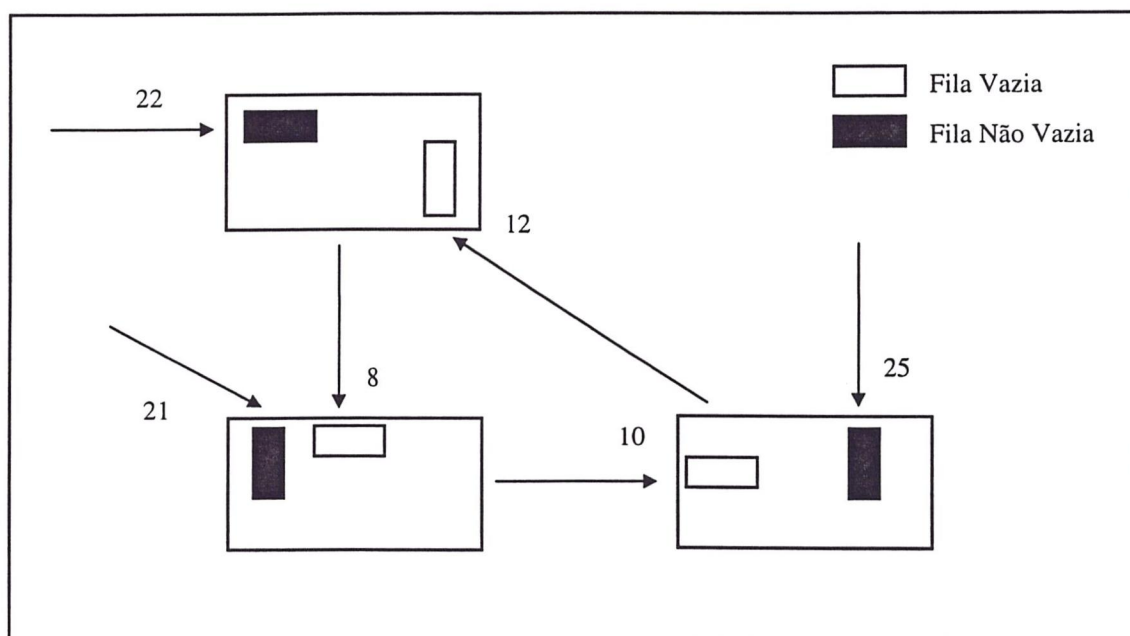


Figura 3.3: Situação de *Deadlock*

Para evitar que a simulação entre em *deadlock*, foi desenvolvido um mecanismo que utiliza mensagens nulas quando não há mensagens reais a serem transmitidas [Fujimoto 1990, Misra 1986]. As mensagens nulas servem somente para sincronização, e não correspondem a qualquer atividade no sistemas físico. São tratadas como qualquer mensagem quando recebidas por um *LP*, ou seja, a recepção de uma mensagem nula faz com que o *LP* atualize seu *LVT*. Assim, uma mensagem nula com marca de tempo T , enviada por um LP_i para um LP_j indica que LP_i não irá enviar mensagens a LP_j entre o valor corrente do relógio e o tempo T . Por essa razão, qualquer mensagem futura de LP_i para LP_j terá marca de tempo superior a T .

O *lookahead* (a quantidade de tempo que um processo pode investigar o futuro) é adicionado ao relógio do canal e transmitido como a marca de tempo das mensagens nulas. Quando um *LP* termina de tratar um evento, envia uma mensagem nula para cada canal de saída, indicando o limite inferior da marca de tempo da próxima mensagem. O receptor da mensagem nula pode então calcular novos limites para os seus canais de saída, enviar essa informação para seus vizinhos, e assim por diante. Isso garante que nenhum *deadlock* pode ocorrer porque alguns *LPs* sempre serão capazes de avançar os relógios de cada canal.

Uma desvantagem do protocolo *CMB* consiste no *overhead* provocado pelas mensagens nulas. Uma variação da abordagem de mensagens nulas consiste em enviar mensagens nulas sob demanda.

Segundo Fujimoto [Fujimoto 1990] a maior desvantagem dos mecanismos conservativos consiste no fato de que estes não podem explorar totalmente o paralelismo disponível na aplicação. Se é possível que um evento E_A possa afetar E_B diretamente ou

indiretamente, as abordagens conservativas devem executar E_A e E_B sequencialmente. Se a simulação é tal que E_A raramente afeta E_B , estes eventos poderiam ser executados concorrentemente na maioria das vezes. Em geral, no pior caso a abordagem conservativa é pessimista, e força a execução sequencial quando não é necessária.

3.2 Mecanismos Otimistas

Os métodos otimistas detectam e recuperam erros de causa e efeito, não os evitam. Em contraste com os mecanismos conservativos, as estratégias otimistas não necessitam determinar quando é seguro executar um evento. Quando ocorre um erro a simulação é restaurada para um estado consistente. Uma vantagem é que esta abordagem permite ao simulador explorar paralelismo em situações onde erros poderiam ocorrer, mas de fato não ocorrem [Fujimoto 1990].

O mecanismo otimista mais conhecido é o *Time Warp*, descrito na seção 3.2.1.

3.2.1 *Time Warp*

O mecanismo *Time Warp* é o protocolo otimista mais conhecido, baseado no paradigma de *Virtual Time* proposto por Jefferson [Jefferson 1985]. O *Time Warp* possui dois componentes principais, o mecanismo de controle local, que garante que os eventos são executados na ordem correta, e o mecanismo de controle global, responsável pelo gerenciamento de memória e detecção do fim da simulação.

A figura 3.4 mostra um processo no mecanismo *Time Warp*. Cada processo contém um relógio local (*Local Virtual Time – LVT*), uma lista de estados (que armazena cópias dos estados recentes do *LP*), uma lista de entrada de mensagens (contendo as mensagens que chegam, em ordem de tempo de recepção) e uma lista de saída de mensagens (contendo cópias das mensagens que o processo enviou). Essas cópias são denominadas antimensagens [Jefferson 1985].

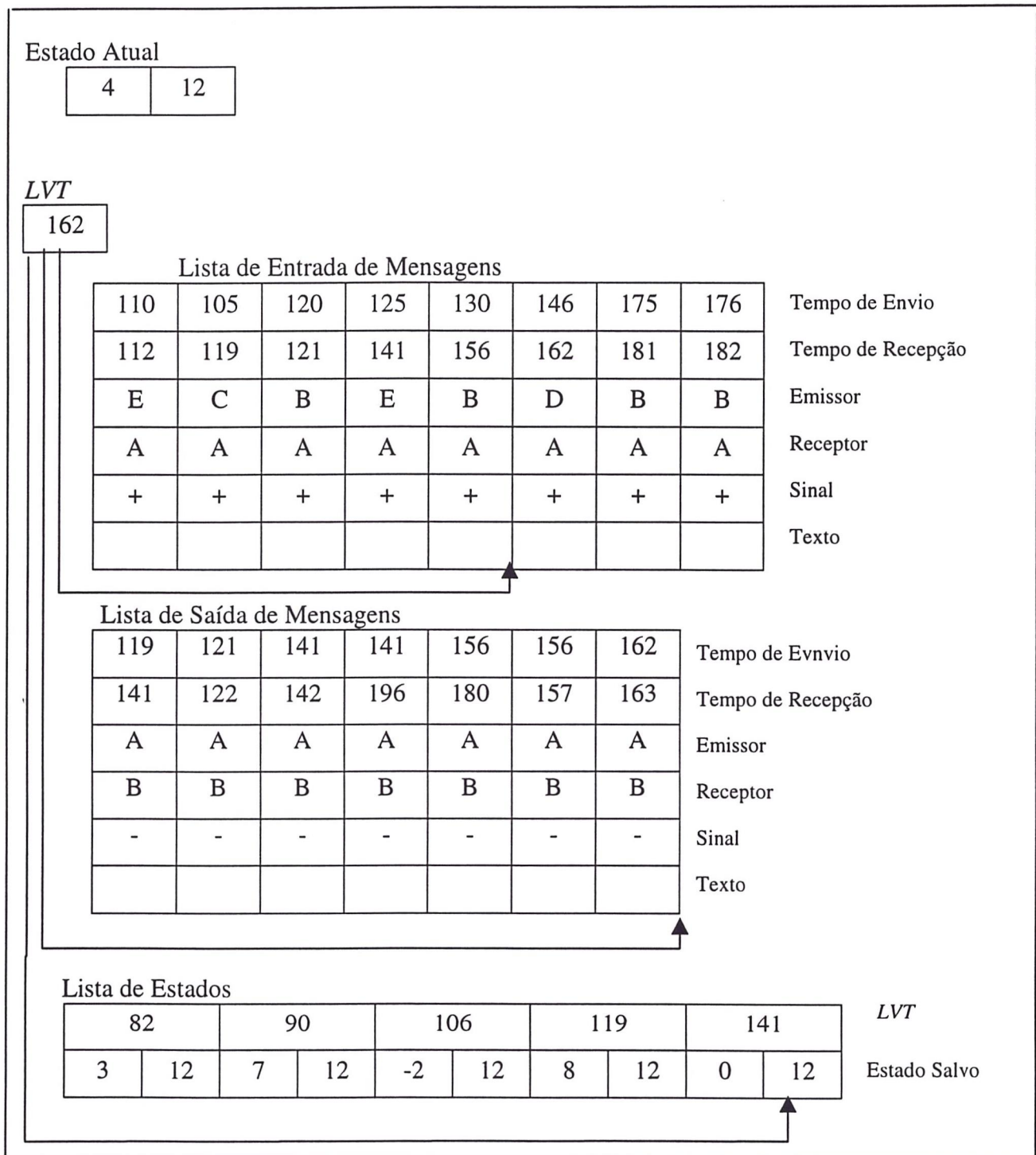


Figura 3.4 Um Processo no *Time Warp*.

O otimismo do mecanismo consiste em permite que os *LPs* executem os eventos até a ocorrência de algum erro. Nesse caso, para retornar a um estado consistente, deve-se desfazer os efeitos de todos os eventos que tenham sido executados prematuramente pelo *LP*. Efetuar *rollback* é a maneira pela qual o *Time Warp* desfaz as computações errôneas que porventura algum *LP* tenha efetuado, por não haver nenhum mecanismo de bloqueio. Para poder restaurar estados, o *Time Warp* necessita guardar os estados dos processos, daí a necessidade da lista de estados. Nesse processo de *rollback*, todas as mensagens enviadas, relativas aos estados que devem ser descartados, também necessitam ter seu envio anulado, e isso é efetuado através do envio de antimensagens, que são versões negativas das mensagens

(positivas) enviadas por um processo. Se um *LP* recebe uma antimensagem que corresponde a uma mensagem positiva que já executou, então também deve retornar para desfazer o efeito da mensagem positiva a ser aniquilada. Repetir este procedimento recursivamente permite que todos os efeitos das computações erradas sejam eventualmente cancelados [Fujimoto 1990].

Cada *LP* na simulação envia e recebe mensagens e antimensagens. Um *LP* executa as seguintes ações ao receber um mensagem:

- A primeira ação a ser efetuada é verificar se a antimensagem correspondente já chegou. Em caso afirmativo, a antimensagem é removida da lista de entrada e ambas são descartadas.
- Se não existe antimensagem correspondente àquela mensagem na fila de entrada, o *LP* verifica a marca de tempo da mensagem, comparando seu valor com o *LVT*.
- Se a marca de tempo é maior ou igual que o *LVT* a mensagem é armazenada na lista de entrada e posteriormente será executada;
- Se a marca de tempo é menor do que o *LVT* ocorreu um erro de causa e efeito e o *LP* sofre um *rollback*.

Caso o *LP* receba uma antimensagem, as seguintes ações são efetuadas:

- Se a antimensagem chegou primeiro do que a mensagem correspondente, é armazenada na lista de entrada;
- Se a mensagem correspondente a antimensagem já chegou ao *LP* e ainda não foi executada, a antimensagem a anula. O *LP* retira a mensagem da lista de entrada e ambas são descartadas;
- Se a mensagem correspondente chegou e já foi executada, o *LP* deve executar um *rollback* para retornar a simulação a um estado seguro.

No mecanismo de controle global, o conceito mais importante é o de *Global Virtual Time* (*GVT*) cujo valor corresponde a menor marca de tempo (menor *LVT*) entre todas as mensagens enviadas e não processadas. Como a necessidade por espaço de memória para salvar estados é grande e nenhum evento com marca de tempo menor do que *GVT* sofrerá *rollback*, os estados salvos até o valor de *GVT* podem ser descartados. Vários algoritmos têm sido propostos para calcular *GVT* [Fujimoto 1990, Fujimoto 1995].

Um problema sério com o mecanismo *Time Warp* é a necessidade de salvar periodicamente o estado de cada processo lógico. O *overhead* de efetuar esta operação pode degradar o desempenho do mecanismo. Além disso, os algoritmos otimistas tendem a utilizar mais memória do que os conservativos [Fujimoto 1990].

O mecanismo *Time Warp* é mais difícil de implementar do que as abordagens conservativas, e depurar tais implementações pode ser dispendioso porque pode exigir análise detalhada de cenários de *rollback* complexos [Fujimoto 1990].

3.2.1.1 Estratégias de Cancelamento

O mecanismo *Time Warp* utiliza Cancelamento Agressivo, ou seja, quando um *LP* efetua *rollback* para o tempo T , são enviadas antimensagens imediatamente, para qualquer mensagem positiva anteriormente enviada com uma marca de tempo maior do que T . No Cancelamento Preguiçoso, os processos não enviam imediatamente as antimensagens para qualquer computação que causa *rollback*, ao invés disso esperam para verificar se a reexecução da computação gera as mesmas mensagens anteriormente enviadas. Nesse caso, não há necessidade de efetuar o cancelamento [Fujimoto 1990, Reiher et al. 1990].

Segundo Fujimoto [Fujimoto 1990], o Cancelamento Preguiçoso pode tanto melhorar quanto prejudicar o desempenho da simulação. Isso porque o mecanismo requer algum *overhead* adicional sempre que um evento é tratado, para verificar se existem antimensagens que possam anulá-lo. Além disso, permite que computações errôneas ocorram, pelo fato de não enviar as imediatamente.

Na estratégia de Cancelamento Dinâmico, uma técnica onde cada *LP* no *Time Warp* decide qual estratégia de cancelamento (Agressivo ou Preguiçoso) vai utilizar. Os *LPs* estudam o passado recente para verificar se o Cancelamento Agressivo teve sucesso ou não, e tomam a decisão tendo por base esse estudo [Radhakrishnan et al. 1997, Wilsey 1997].

No Cancelamento Experimental uma única antimensagem é utilizada para cancelar um grupo de mensagens [Kalanter 1997].

3.2.1.2 Estratégias para Salvamento de Estados

O algoritmo utilizado para salvamento de estados constitui uma parte fundamental de qualquer sistema onde computações errôneas devam ser desfeitas e o estado do sistema restaurado. A tarefa de salvar estados no *Time Warp* é complexa, exige mecanismos que ajudem a minimizar a utilização do espaço de memória disponível. Pode ser implementada pelos seguintes algoritmos [Rönngrén et al. 1996, Sköld; Rönngrén 1996]:

- *Copy state saving*: o estado do *LP* é salvo a cada vez que um evento é executado;
- *Sparse state saving*: o estado do *LP* é salvo periodicamente, em intervalos fixos durante toda a simulação ou que se adaptam ao comportamento da simulação;
- *Incremental state saving*: são salvas somente as partes do estado que foram modificadas.

4. Limitação do Otimismo no *Time Warp*

Sem uma escolha clara entre protocolos otimistas e conservativos, os pesquisadores têm estudado variações híbridas que implementam uma abordagem intermediária entre os extremos conservativo e otimista. Intuitivamente, esses protocolos utilizam uma forma de otimismo calculado, de forma que podem explorar os benefícios de otimismo (explorar mais paralelismo) sem alguns de seus problemas (*overhead* causado por *rollbacks* e consumo de memória) [Reiher; Jefferson 1989], ou então adicionam mais otimismo em alguma abordagem conservativa. Um dos primeiros a estudar uma estrutura para protocolos híbridos foi Reynolds [Reynolds 1989], definindo os conceitos de agressividade e risco. Um protocolo

agressivo pode executar eventos fora de ordem de marca de tempo o que pode levar a *rollbacks*. Um protocolo com risco, além de ser agressivo, pode enviar mensagens que tenham sido geradas baseado em processamento agressivo, e tais mensagens podem ter que ser canceladas. O protocolo *Time Warp* explora ao máximo os conceitos de agressividade e risco. Por outro lado, os protocolos conservativos não empregam agressividade ou risco. Segundo Reynolds, agressividade e risco podem ser consideradas duas variáveis de projeto importantes, que podem influenciar o desempenho dos protocolos para simulação paralela e distribuída [Das 1996].

Na literatura de simulação paralela, estão descritos vários mecanismos para sincronização que são basicamente variantes dos mecanismos *CMB* e *Time Warp*. A quantidade de mecanismos leva à necessidade de uma classificação que permita estudá-los de uma mais organizada.

Das [Das 1996] e Srinivasan [Srinivasan; Reynolds 1995a, Srinivasan; Reynolds 1995b] apresentam em seus trabalhos classificações que não abrangem todos os mecanismos já desenvolvidos. Das [Das 1996] divide os protocolos em dois grandes grupos, híbridos e adaptativos, e considera tanto os protocolos que introduzem otimismo em protocolos conservativos quanto os protocolos que introduzem idéias conservativas em protocolos otimistas. Segundo o autor, alguns protocolos são difíceis de classificar, e por isso apresenta uma classificação baseada mais no espírito do protocolo. Uma das classificações de Srinivasan trata apenas dos protocolos que limitam o otimismo no *Time Warp*, e está baseada no critério que os mesmos utilizam, e a outra define uma classe de protocolos adaptativos, isto é, que ajustam seu comportamento dinamicamente, de acordo com o modelo.

É interessante destacar que Steinman [Steinman 1991] descreve o protocolo utilizado no *SPEEDES (Breathing Time Buckets)* como moderado, um meio termo entre otimista e conservativo.

Classificação de Das:

- Protocolos híbridos:
 - Protocolos que adicionam otimismo a protocolos conservativos: *Simulação Especulativa, SRADS, Breathing Time Buckets, Filtered Rollback*;
 - Protocolos que tentam controlar o otimismo no *Time Warp*: *Moving Time Windows, Bounded Time Warp, MIMDIX, Breathing Time Warp, Composite ELSA, Local Time Warp, Wolf, Filter*;
- Protocolos adaptativos, que se ajustam entre a abordagem otimista e a abordagem conservativa:
 - Gerais: *Moving Time Windows, MIMDIX*;
 - Baseados no estado local: *Penalty-based Throttling, Adaptive Time Warp, Protocolo probabilístico baseado em custos, CO-OP, Probabilistic Distributed Discrete Event Simulation Protocol, Local Adaptive Protocol, PADO, Adaptive Bounded Time Window*;
 - Baseados no estado global: *Adaptive Memory Management Protocol, protocolos Near Perfect State Information (NPSI), Cancelback Protocol*.

Classificação de Srinivasan para os protocolos que limitam o otimismo no *Time Warp*:

- Baseados em janelas: somente os eventos dentro da janela são executados. Isso garante que todos os *LPs* permanecem próximos uns aos outros, no tempo simulado, e não podem ocorrer *echoing* e *rollbacks* em cascata. Exemplos: *Moving Time Windows*, *Window based Throttling*, *Bounded Time Warp*, *Breathing Time Warp*;
- Baseados no espaço: utilizam limites espaciais ao invés de limites temporais (janelas). Em geral, os processadores são divididos em grupos, e *Time Warp* é utilizado internamente. Os grupos interagem sem risco, de forma conservativa. Exemplos: *Local Time Warp*, *SRADS*, *Breathing Time Warp*;
- Baseados em penalidade o passado recente é utilizado para prever o futuro próximo. Baseado em seu comportamento recente, alguns *LPs* são penalizados (bloqueados) e outros são favorecidos (podem continuar a execução). Exemplo: *Penalty-based Throttling*;
- Baseados em conhecimento: informação sobre *rollback* é utilizada para restringir a propagação de computação provavelmente incorreta. Exemplos: *Filter*, *WOLF*;
- Probabilísticos: um processo especial envia, periodicamente, mensagens de sincronização a todos os *LPs*. Exemplo: *MIMDIX*;
- Adaptativos e baseados no estado: existe um protocolo que limita, de forma adaptativa, o consumo de memória, e conseqüentemente limita o otimismo. Já os protocolos adaptativos *NPSI* (seção 4.14) limitam o otimismo e conseqüentemente limitam o consumo de memória.

Classificação de Srinivasan para protocolos adaptativos (que se adaptam às mudanças no sistema):

- *Asynchronous Adaptive Waiting Protocols (AAWP)*: controlam o otimismo introduzindo atrasos entre as execuções dos eventos. Os mecanismos que utilizam janelas globais não são *AAWP* porque a janela global força todos os *LPs* a sincronizarem. Exemplos: *Penalty-based Throttling*, *Adaptive Time Warp*, *Local Adaptive Protocol*, protocolos adaptativos *NPSI*, *Breathing Time Warp*, *UDS*.

Alguns protocolos desenvolvidos posteriormente à divulgação das duas classificações não foram considerados, como é o caso do protocolo *Tentative Time Warp*. A taxonomia apresentada por Spolon [Spolon et al. 1999] permite uma melhor visualização dos mecanismos otimistas, consistindo em uma abordagem genérica e hierárquica.

Reynolds [Reynolds 1989] define adaptatividade de um protocolo para simulação paralela e distribuída como a capacidade de um protocolo modificar suas variáveis de projeto durante a simulação. Segundo Srinivasan [Srinivasan; Reynolds 1995a], vários protocolos que limitam o otimismo não são adaptativos por essa definição. Para que os *LPs* tomem decisões dinâmicas baseados no estado do sistema, necessitam de informações de estado quase perfeitas a baixo custo, o que tem sido o maior obstáculo para o estudo de protocolos *NPSI* (*Near Perfect State Information*, seção 4.14).

Ainda segundo Srinivasan [Srinivasan; Reynolds 1995a], esses protocolos apresentam melhor desempenho que o *Time Warp*, mas em situações de teste específicas e seu desempenho, de forma geral, é limitado. Isso é devido ao fato de que o critério para limitar o otimismo (tamanho da janela, tamanho do grupo, penalidade, probabilidades, etc) é pré-determinado e a decisão para limitar o otimismo é baseada somente na história local. Porém, esses protocolos podem eliminar a necessidade de esquemas complexos para gerenciamento de memória.

Esta seção descreve os principais mecanismos que alteram o *Time Warp* de forma a limitar seu excesso de otimismo. Os protocolos *Lock-Step Time-Driven Synchronization* e *Time Buckets Synchronization* não são classificados como otimistas ou como protocolos que limitam o otimismo no *Time Warp*, porém são a base para os protocolos *Breathing Time Buckets* e *Breathing Time Warp* e por isso estão descritos nos itens 4.1 e 4.2. Todos os demais introduzem alguma forma de limitação para controlar o excesso de otimismo no *Time Warp*.

4.1 Lock-Step Time-Driven Synchronization

Abordagem mais simples para sincronização, executa todas as computações em intervalos regulares com um incremento de tempo fixo. Por exemplo, o processamento poderia ocorrer nos tempos 0.0, 7.5, 15.0, 22.5, ... até alcançar o final da simulação. Cada *LP* executa seu processamento no tempo global de simulação corrente e espera os demais *LPs*. Quando todos completam seu processamento nesse tempo, trocam mensagens e incrementam seus relógios [Steinman 1995].

Desvantagens:

- Os incrementos de tempo com valores fixos não modelam com fidelidade o mundo real;
- A diminuição do valor incremento de tempo aumenta o *overhead* devido a necessidade de mais sincronizações.

4.2 Time Buckets Synchronization

Uma das abordagens mais simples para simulação paralela, desenvolvida em 1990, utiliza o princípio de causa e efeito (figura 4.1) [Steinman 1992]. A simulação é executada em ciclos com períodos de duração T , onde T é o intervalo mínimo conhecido entre eventos e os eventos que estes podem gerar [Steinman 1992, Steinman 1993].

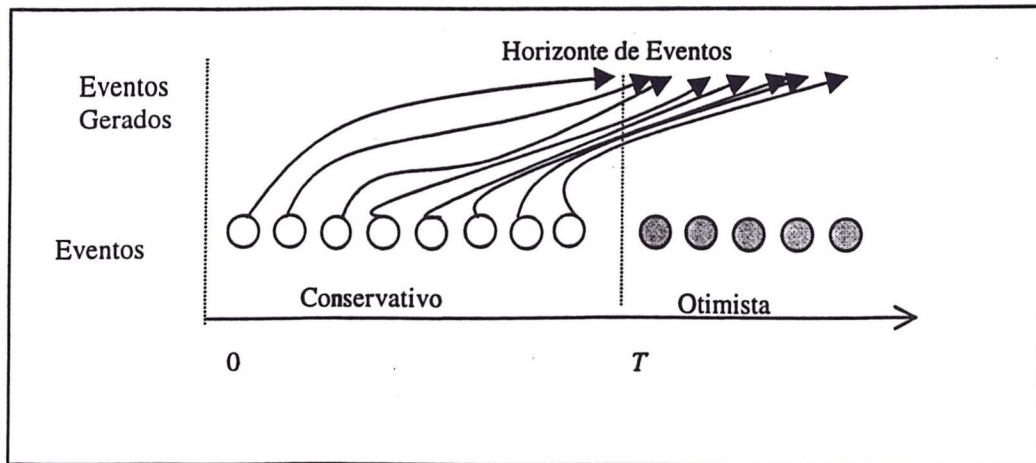


Figura 4.1: Princípio de Causa e efeito.

Os nós devem sincronizar com todos os demais no final de cada ciclo, e depois disso todos avançam seu tempo de simulação em T . O mecanismo garante que cada nó não irá receber uma mensagem com marca de tempo fora do ciclo corrente.

O princípio de causa e efeito permite que os eventos até horizonte de eventos sejam executados, de forma conservativa, em paralelo. O horizonte de eventos é definido como o ponto no tempo no qual um evento, para ser executado, tenha uma marca de tempo menor do que a marca de tempo do evento mais recente gerado no ciclo corrente. Podem ocorrer erros de simulação se os eventos forem executados, de forma otimista, além do horizonte de eventos [Steinman 1992].

Vantagem:

- Exige pouco *overhead* para sincronização [Steinman 1993].

Desvantagens:

- O ciclo deve ser grande o suficiente para que cada nó processe eventos em quantidade adequada para tornar a simulação distribuída eficiente;
- Segundo Steinman [Steinman 1993, Steinman 1995], não é uma estratégia de sincronização de propósito geral adequada e não contém escalabilidade;
- Quando um *LP* tem mais trabalho a fazer do que outro, a eficiência da simulação depende muito do balanceamento de carga.

4.3 Breathing Time Buckets (BTB)

Algoritmo utilizado no ambiente *SPEEDES* (*Synchronous Parallel Environment for Emulation and Discrete Event Simulation*) [Steinman 1991, Steinman 1992] e baseado em janelas. Utiliza processamento otimista com *rollback* local. Não necessita de antimensagens, como em outros ambientes com janelas (*MTW*, seção 4.9) [Steinman 1993]. Resolve os problemas de instabilidade exibidos no *Time Warp*, pois processa os eventos de forma otimista, mas as mensagens geradas por eventos são liberadas somente quando se tem certeza de que são válidas [Steinman 1995]. Mistura os conceitos de *Time Warp* e o algoritmo *Fixed*

Time Buckets. Diferente das abordagens conservativas, o método permite conectividade total entre os objetos da simulação. O conceito essencial da sincronização proposta é o princípio de causa e efeito, como mostrado na figura 4.2 [Steinman 1993].

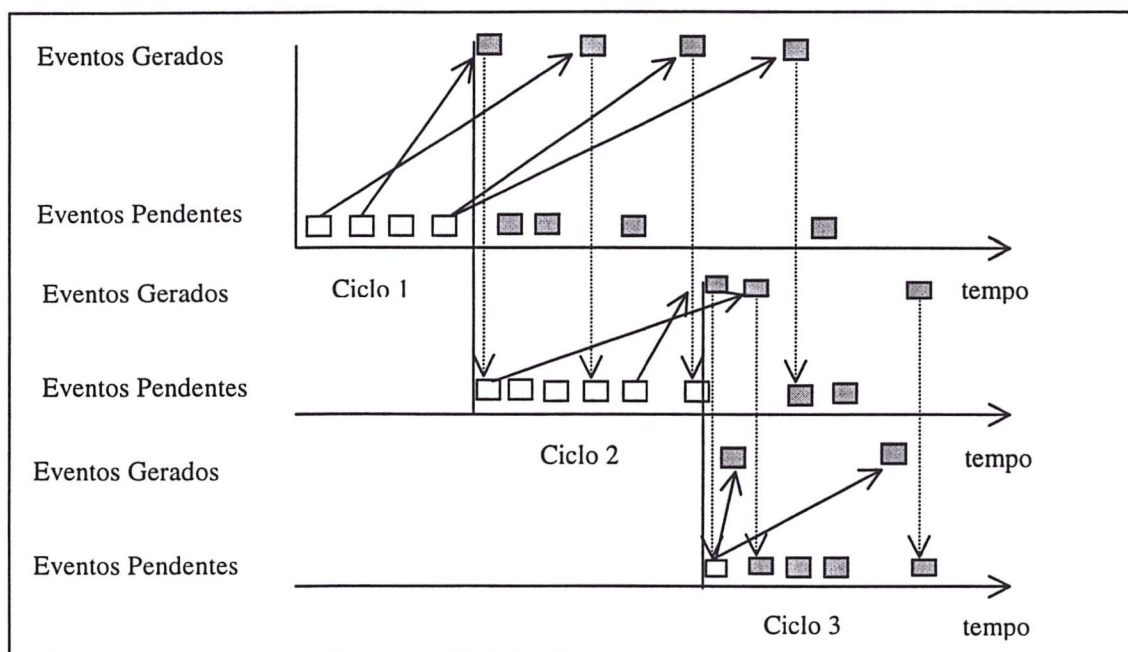


Figura 4.2: Um Exemplo de Processamento de Eventos em Ciclos definidos pelo Horizonte de Eventos.

BTB processa os eventos em ciclos (como no *Fixed Time Buckets*), que se adaptam a uma largura ótima determinada pelo horizonte de eventos. Isso implica que em cada ciclo o número máximo de eventos corretos é processado, e o princípio de causa e efeito permite que os eventos em cada ciclo sejam executados em paralelo [Steinman 1993].

O horizonte de eventos (a marca de tempo do primeiro evento gerado no ciclo corrente) determina o limite do próximo ciclo. Processar eventos além desse limite pode levar a erros de causa e efeito e à necessidade de *rollback*. Assim como no caso do *LVT* no *Time Warp*, define-se horizonte de eventos local e global. O processamento de eventos de forma otimista é utilizado para determinar o horizonte de eventos global. As mensagens são liberadas somente depois que o horizonte de eventos global é determinado, ou seja, são enviadas de forma conservativa, sem risco. Assim, não são necessárias antimensagens, e *rollback* simplesmente envolve restaurar o estado do *LP*, e descartar as mensagens que foram geradas de forma incorreta [Steinman 1992].

No final de cada ciclo, é executada a coleta de lixo, para liberar espaços de memória utilizados para salvar estados que não são mais necessários.

Para determinar quando o último nó ultrapassou o horizonte de eventos, e assim o fim do ciclo, existem algumas técnicas:

- Utilizar uma função de sincronização não bloqueante. Cada nó chama essa função assim que percebe que atingiu o horizonte de eventos, e continua seu processamento. Quando o último nó chama a função, todos sincronizam e executam coleta de lixo;

- Usar *broadcast* assíncrono para informar os outros nós da existência de um limite superior para o horizonte de eventos. Um nó só envia *broadcast* assíncrono se acredita que seu horizonte de eventos local é o horizonte de eventos global. Essa decisão é baseada no valor do horizonte de eventos local e nas mensagens de *broadcast* assíncrono que o nó recebe de outros nós.

Vantagem:

- Como não utiliza antimensagens, não apresenta *overhead* para manipulá-las, como no *Time Warp*.

Desvantagens:

- Para que o mecanismo seja eficiente, exige que em um ciclo sejam processados eventos suficientes. Entretanto, sempre processa pelo menos a mesma quantidade de eventos por ciclo como o mecanismo *Fixed Time Buckets* [Steinman 1992];
- Não apresenta boa escalabilidade para simulações que exigem pouca interação entre os objetos da simulação, ou seja, para simulações que não processam evento suficientes por ciclo, muitas sincronizações serão necessárias e haverá perda de escalabilidade. Existe um modelo analítico que prediz quantos eventos podem ser processados, em média, por ciclo, para um comportamento de equilíbrio [Steinman 1995];
- Necessita de técnicas sofisticadas para determinar quando o último *LP* ultrapassou o horizonte de eventos [Ferscha 1995a].

4.4 *Breathing Time Warp (BTW)*

A motivação para esse protocolo é o fato de que o *Time Warp* é extremamente otimista, como por exemplo em simulações onde a granulosidade dos eventos é muito pequena (o *overhead* de comunicação é significativo, cada mensagem que é enviada ou recebida gasta tempo na simulação). Nesses casos, pode ser importante enviar somente as mensagens que têm uma boa chance de serem válidas (reduzir o risco de forma que não ocorram explosões de antimensagens) [Steinman 1995]. Já o algoritmo *Breathing Time Buckets* [Steinman 1992, Steinman 1995], no outro extremo, somente envia as mensagens quando sabe que são válidas. Nesse sentido pode ser muito conservativo.

Assim, a principal motivação para o algoritmo *BTW* é a premissa de que a probabilidade de um evento processado sofrer *rollback* tende a aumentar com a distância (em termos do número de eventos) em relação ao valor de *GVT* (*Global Virtual Time*). Assim, os eventos próximos a *GVT* (em termos de número de eventos) tendem a ser processados corretamente.

Essa é a estratégia básica do *BTW*, liberar as mensagens próximas a *GVT*, mas não liberar imediatamente as mensagens geradas por eventos que estão distantes de *GVT*.

O mecanismo é uma mistura de *Time Warp* e *BTB* [Steinman 1992, Steinman 1995]. Os eventos podem ser processados de forma conservativa até $GVT + T$ tal que o *overhead* de salvar estados pode ser eliminado para aqueles eventos (T é a distância entre eventos e os eventos que eles geram, ou seja, *lookahead*).

O mecanismo é composto pelas fases:

1) Fase *Time Warp*

Inicia o algoritmo. Os primeiros N_1 eventos processados além de GVT têm suas mensagens enviadas imediatamente. Como N_1 não é muito distante de GVT , isso significa que provavelmente tais mensagens não irão causar *rollback*. Quando os N_1 eventos são processados, inicia-se a fase 2.

2) Fase *Breathing Time Buckets*

Nesta fase os eventos são processados mas as mensagens não são liberadas. A marca de tempo mínima de todas as mensagens não enviadas é monitorada como uma estimativa do horizonte de eventos, de acordo com o algoritmo *Breathing Time Buckets* (seção 4.3).

Nesta fase, podem existir mensagens e antimensagens nos canais de comunicação, e possivelmente alguma delas pode ser relativa a eventos processados na fase 1. Nesse caso, ocorre o chaveamento para a fase 1, apenas para aquele evento.

Quando o horizonte de eventos é alcançado, todos os nós param seu processamento e iniciam o cálculo de GVT .

3) Fase GVT

Nesta fase o valor de GVT é determinado. Esse é um dos pontos (dos mecanismos otimistas) que têm merecido vários estudos. Um dos problemas em se determinar esse valor é garantir que não existam mensagens circulando pelos canais de comunicação.

4) Fase de Coleta de Lixo

As mensagens que não foram liberadas são enviadas sincronamente (aquelas com marca de tempo menor ou igual a GVT). Esta fase faz então a liberação do espaço de memória ocupado com estados salvos durante a fase 1.

Vantagens:

- Ao liberar somente as mensagens de N_1 eventos além de GVT , evita avalanches de antimensagens, ou seja, melhora *Time Warp* no sentido de que resolve os problemas de instabilidade observados no mesmo;
- O desempenho é limitado somente pela quantidade de paralelismo da simulação, não pelo algoritmo de sincronização [Steinman 1995];
- Reduz a taxa de risco do *Time Warp* e é melhor do que *BTB* pois esse pode ser muito conservativo, na tentativa de eliminar antimensagens (porque pode ser ineficiente se o número de eventos por ciclo é pequeno).

Desvantagens:

- O usuário deve determinar o valor de N_1 . Segundo Steinman [Steinman 1995], o valor ótimo depende do hardware e da granulosidade dos eventos, e não é função do tempo de simulação;

- Para efetuar o cálculo de *GVT* precisa garantir que não há mensagens circulando entre os *LPs*;
- A fase 4 é complexa, com manipulação de duas listas de eventos.

4.5 Local Time Warp (LTW)

Combina de forma hierárquica o algoritmo *Conservative Time Windows* [Ayani; Rajaei 1994] e *Time Warp*. O sistema simulado consiste em um número de subsistemas (*grupos*), compostos por vários *LPs* [Rajaei et al. 1993]. Dentro dos *grupos*, utiliza *Time Warp* para explorar melhor o paralelismo da aplicação. Para impedir que *rollbacks* de um cluster afetem outro, utiliza *Conservative Time Windows* [Ayani; Rajaei 1994] para a sincronização entre os mesmos.

Os *LPs* de um *grupo* podem transmitir mensagens diretamente uns aos outros. Já os *grupos* são interconectados através de canais. Cada *grupo* possui 2 *Gateway Processes (GPs)* que executam a comunicação entre os mesmos. Cada *LP* possui uma identificação única, o que permite verificar, no endereço do *LP* destinatário de uma mensagem, se o mesmo é interno ou externo ao *grupo*. Cada *grupo* possui um *Group Virtual Time (CVT)*, semelhante ao *GVT* do *Time Warp*. A definição de *virtual time* também se estende aos *Gateway Process (Input Virtual Time e Output Virtual Time)* [Rajaei et al. 1993].

Entre os *LPs*, dentro de cada *grupo*, é utilizado o mecanismo *Time Warp* e o *GVT* é substituído pelo *Group Virtual Time (CVT)*. As mensagens destinadas a *LPs* externos ao *grupo* são passadas automaticamente para o *Output Gateway*, e são mantidas ali até que o *CVT* do *grupo* se torne maior ou igual a suas marcas de tempo. Entre os *grupos*, a relação $IVT \geq CVT \geq OVT$ é utilizada por um algoritmo conservativo para garantir que nenhum *rollback* ocorrendo em um *grupo* possa se propagar e disparar reações em cadeia em outro *grupo*. Ou seja, não é permitido que uma mensagem chegue ao *IGP* de um *grupo* com marca de tempo menor que seu *IVT* [Rajaei et al. 1993].

Vantagens:

- Previne *rollbacks* em cascata, porém preserva a flexibilidade do *Time Warp*;
- O protocolo é geral o suficiente para permitir a utilização de outros protocolos de sincronização entre os *grupos* [Rajaei et. al 1993];
- É possível efetuar o cálculo do *CVT* em cada *grupo* de forma mais *frequente*, e portanto pode ter melhor controle sobre o espaço de memória necessário para guardar estados;
- Permite a criação e encerramento dinâmico de *LPs* dentro dos *grupos*, como no *Time Warp*. Também permite a criação e encerramento dinâmico de *grupos*;
- Os *rollbacks* dentro de um *grupo* não afetam outros *grupos*, porque só saem dos mesmos mensagens seguras;

Desvantagens:

- O trabalho de dividir os *LPs* em *grupos* é deixado para o usuário;

- Devido ao pequeno número de *LPs* nos *grupos* (em comparação ao *Time Warp*), o cálculo do *CVT* em cada um pode tornar-se mais oneroso do que o cálculo de *GVT* no *Time Warp*.

O algoritmo *Conservative Time Windows* (CTW) [Ayani; Rajaei 1992, Ayani; Rajaei 1994] divide o sistema a ser simulado em n sub-sistemas disjuntos, cada um representado por um objeto. O algoritmo identifica uma janela de tempo para cada um, de forma que os eventos pertencentes a janelas diferentes sejam independentes e possam portanto ser executados concorrentemente. O tamanho de cada janela é calculado em cada iteração do algoritmo utilizando características diferentes do sistema sendo simulado, o que pode implicar em janelas com tamanhos diferentes nos objetos.

Utiliza o *Three Phase Algorithm*, que diferente do mecanismo *CMB*, é livre de *deadlocks* [Ayani; Rajaei 1992]:

repeat

1) *Fase Inicial*

Marque o primeiro evento de cada objeto como um candidato para avaliação concorrente

Barreira

2) *Fase de Controle de Concorrência*

Identifique os eventos independentes entre os marcados

Barreira

3) *Fase de Avaliação*

Execute os eventos independentes

Barreira

Until (End of Simulation).

4.6 *Bounded Time Warp*

O mecanismo *Bounded Time Warp* divide o tempo de simulação em intervalos de tamanhos iguais, e é similar ao *Moving Time Windows* (seção 4.9). Porém, ao invés de arrastar as janelas concorrentemente com o processamento de eventos, todos os eventos dentro do intervalo corrente são simulados antes que o próximo evento seja iniciado [Das 1996, Rajaei et al. 1993].

O ponto central desse método consiste em um algoritmo que pode detectar quando todos os objetos (*LPs*) estão inativos, isto é, não existem mensagens não processadas dentro da janela corrente. O algoritmo utiliza um método de passagem de *token* de duas fases. Um *LP* (o mestre, inicialmente) coloca o *token* no final de sua fila de eventos. Na frente do *token* existem mensagens para serem processadas. As mensagens que chegam depois são inseridas na frente do *token*, de forma que esse fica sempre no final da fila. Quando o *token* é processado, significa que o *LP* está inativo. O *LP* corrente então passa o *token* para o próximo *LP* e assim sucessivamente (os *LPs* formam um anel lógico) [Turner; Xu 1992a].

Vantagens [Turner; Xu 1992b]:

- Apresentou de 27% a 87% de melhoria em relação ao mesmo modelo em *Time Warp*;

- Não podem ocorrer *rollbacks* com amplitude ilimitada, porque a distância mínima de *rollback* não pode exceder um valor limite.

Desvantagens [Turner; Xu 1992a]:

- Como determinar o limite (*bound*);
- Se o limite é pequeno (nos experimentos, menor do que 50), o *overhead* de passagem de mensagens é grande.

4.7 Filter

Filter é um protocolo otimista baseado em conhecimento condicional, voltado para melhorar o desempenho de algoritmos otimistas. Acelera o cancelamento de computações errôneas ao custo de manter algum estado adicional. Cada mensagem carrega uma lista de suposições ou condições que descrevem a classe de eventos que pode causar seu cancelamento. Ao receber uma mensagem desse tipo, um *LP* divulga, através de *broadcast*, informação sobre a mensagem recebida, utilizando uma mensagem *rollback-info*. Isso permite filtrar qualquer mensagem errônea que chega e também efetuar *rollback* para um ponto anterior se necessário [Prakash; Subramanian 1991, Prakash; Subramanian 1992].

Vantagens:

- O cancelamento de eventos é feito rapidamente, através de uma única mensagem [Prakash; Subramanian 1992];
- Evita o processamento de mensagens que poderiam eventualmente ser aniquiladas por antimensagens em trânsito (baseado na lista de suposições que a mesma carrega);
- Segundo Prakash [Prakash; Subramanian 1991], *Filter* pode aumentar o *overhead* por mensagem, mas reduz o número total de mensagens enviadas, o número de mensagens processadas e o número de *rollbacks*.

Desvantagens:

- O custo de cancelamento de eventos implica em um alto *overhead* por mensagem, na forma da lista de suposições que cada uma carrega. Essa lista pode se tornar muito grande para simulações com um grande número de *LPs* [Prakash; Subramanian 1991];
- Utiliza antimensagens como no *Time Warp*.

4.8 SFilter (Single-hop Filter)

Variação do mecanismo *Filter*. Cada *LP* transmite apenas suas suposições locais ao invés da lista inteira de suposições (como ocorre em *Filter*), e apenas para seus vizinhos. O cancelamento de mensagens não é feito através de *broadcast* global, mas local. Essa limitação da propagação da informação pode reduzir o *overhead* por mensagem.

Vantagem:

- Reduz o número total de *rollbacks* e uso das mensagens *rollback-info* para cancelar múltiplas mensagens.

Desvantagem:

- Alto *overhead* por mensagem, na forma da lista de suposições que cada uma carrega. As listas podem se tornar muito longas para simulações contendo um grande número de processos lógicos [Prakash; Subramanian 1992].

4.9 Moving Time Windows (MTW)

Desenvolvido em 1989 e projetado para dar suporte a abordagem para simulação orientada a objetos, utiliza janelas que se movem no tempo de simulação. Uma janela de tempo global, que é dinamicamente ajustada, é atribuída aos *LPs* [Ayani; Rajaei 1994]. Dois eventos e e e' com tempos de ocorrência $ot(e)$ e $ot(e')$ podem ser simulados em paralelo somente se $|ot(e) - ot(e')| < \Delta$ [Ferscha 1995a]. A janela, para limitar a computação otimista, tem tamanho Δ pré-definido e é arrastada através do tempo virtual [Rajaei et al. 1993].

Os objetos de simulação (*LPs*) podem selecionar uma das três estratégias de correção de erros, baseados no tipo de erro e da mensagem. Assim, tenta evitar o *overhead* associado com a abordagens baseadas em *rollback* [Sokol; Stucky 1990].

Esse protocolo deu origem aos trabalhos que procuram limitar a ocorrência de *rollbacks* e o consumo de espaço em memória no *Time Warp*, limitando assim seu otimismo [Das 1996].

Vantagem:

- Favorece modelos de simulação com uma variação pequena das distâncias de ocorrência dos eventos relativo ao tamanho da janela.

Desvantagens:

- Como determinar o valor de Δ . Janelas muito pequenas limitam os *rollbacks* porém exploram pouco o paralelismo, enquanto janelas grandes, apesar de explorarem o paralelismo da aplicação, implicam em maior ocorrência de *rollbacks*;
- Supõe de forma implícita que os tempos de ocorrência dos eventos são distribuídos uniformemente [Ferscha 1995a];
- Não leva em conta o fato de que eventos além do limite superior da janela também podem ser executados corretamente (visão otimista) [Ferscha 1995a].

4.10 Adaptive Time Warp Concurrency Control Algorithm (ATW)

Suspende temporariamente o processamento dos eventos, ao observar um certo número de violações de causa e efeito, ou seja, pára o avanço do *LVT* por um período de tempo denominado *Blocking Window (BW)*. Somente os eventos com marcas de tempo dentro da janela podem ser executados [Ferscha 1995b]. O tamanho da *BW* adaptativa é determinado baseado no mínimo de uma função que descreve a computação desperdiçada em termos do tempo gasto em modo bloqueado (conservativo) ou modo de recuperação de falha (mecanismo de *rollback* do *Time Warp*) [Ball; Hoyt 1990, Ferscha 1995a].

O valor da *BW* é atualizado periodicamente durante a simulação. Se $BW = 0$, não há período de bloqueio e o mecanismo se comporta como *Time Warp*. Se $BW = \infty$ (o bloqueio é certo) o mecanismo se comporta como uma abordagem conservativa [Rajaei et al. 1993].

Desvantagem:

- Cálculo do tamanho da *Blocking Window* é complexo.

4.11 Probabilistic Distributed Discrete Event Simulation Protocol

A motivação desse protocolo é reduzir o número de mensagens de sincronização, evitando ao máximo a ocorrência de *rollbacks*. Consiste em um esquema de janelas cujos tamanhos variam dinamicamente, dependendo do paralelismo inerente ao modelo (*local virtual time windows -LVTW*). O mecanismo adapta probabilisticamente o tamanho da *LVTW*, de acordo com o processo de chegada de eventos externos observado. Ou seja, o mecanismo enfraquece a regra conservativa de bloquear um *LP* até que seja seguro de forma que bloqueia somente se não exista chance de efetuar trabalho correto [Ferscha; Chiola 1994].

Segundo o autor, o protocolo reduz a interação entre os *LPs* causadas pelos *rollbacks*, em troca de certa complexidade.

Vantagens [Ferscha 1995a]:

- O protocolo explora localmente a probabilidade da mensagem que chega ser um *straggler*
- Leva em conta as características arquiteturais da plataforma (como velocidade da CPU e latência de comunicação, é adaptativo.

4.12 Probabilistic Direct Optimism Control (PADOC)

Cada *LP* monitora o progresso do *LVT* em cada canal de chegada, ou seja, registra as marcas de tempo das mensagens assim que elas chegam. Do padrão de chegada observado, cada *LP* formula um teste de hipótese sobre a marca de tempo da próxima mensagem esperada, e - relacionado à confiança estatística no valor previsto - adapta-se, por meio de *throttling* (controle de fluxo) a um comportamento de sincronização que presumivelmente é o melhor entre *CMB* e *Time Warp*. Bloqueia temporariamente o processamento dos eventos, evitando a geração e envio de mensagens em estados para os quais é provável que tenham que ser *rolled back*. Por outro lado, todos os *LPs* tendem a esperar, com grande probabilidade, a chegada de mensagens que podem levar a erros de causa e efeito, exatamente para evitar esse

tipo de violação. Uma análise estatística da estória das chegadas de mensagens é utilizada para fazer previsões para as marcas de tempo das mensagens futuras, permitindo a cada *LP* adaptar localmente seu comportamento em relação à estratégia de sincronização mais eficiente [Ferscha 1995a]. Combina características do *Probabilistic Distributed Discrete Event Simulation Protocol* e *Adaptive Time Warp* [Ferscha 1995d].

Vantagem:

- Independente da razão da comunicação e da velocidade de computação da plataforma, a política de sincronização é ajustada automaticamente, na forma mais adequada para explorar o paralelismo do modelo de simulação.

4.13 MIMDIX

Segundo Madisetti [Madisetti et al. 1992], *MIMDIX* 1.0 é um sistema operacional de segunda geração para dar suporte a simulações de sistemas dinâmicos abertos e fechados, em computadores paralelos. A primeira geração de simuladores paralelos para redes fechadas foi o *Time Warp*, desenvolvido na *UCLA* e no *JPL/Caltech*. Ainda segundo o autor, nenhum dos ambientes até *MIMDIX* era capaz de se adaptar à natureza dinâmica da computação.

Em um ambiente de sincronização probabilístico como o *MIMDIX*, em pontos discretos do tempo cada processador envia (ou não), com certa probabilidade, uma mensagem de re-sincronização aos demais processadores. Sua marca de tempo é a marca de tempo local, e todas as mensagens na lista de eventos do receptor com marca de tempo acima desse valor são descartadas. Entre esses pontos de sincronização, as antimensagens preservam a corretude da simulação. Isso mantém no *MIMDIX* características de mecanismos conservativos e de mecanismos otimistas para sincronização de simulação distribuída (a sincronização probabilística não bloqueia processos, não utiliza nenhum mecanismo de janelas). [Madisetti et al. 1992]

A figura 4.3 mostra o ambiente *MIMDIX* [Madisetti et al. 1992].

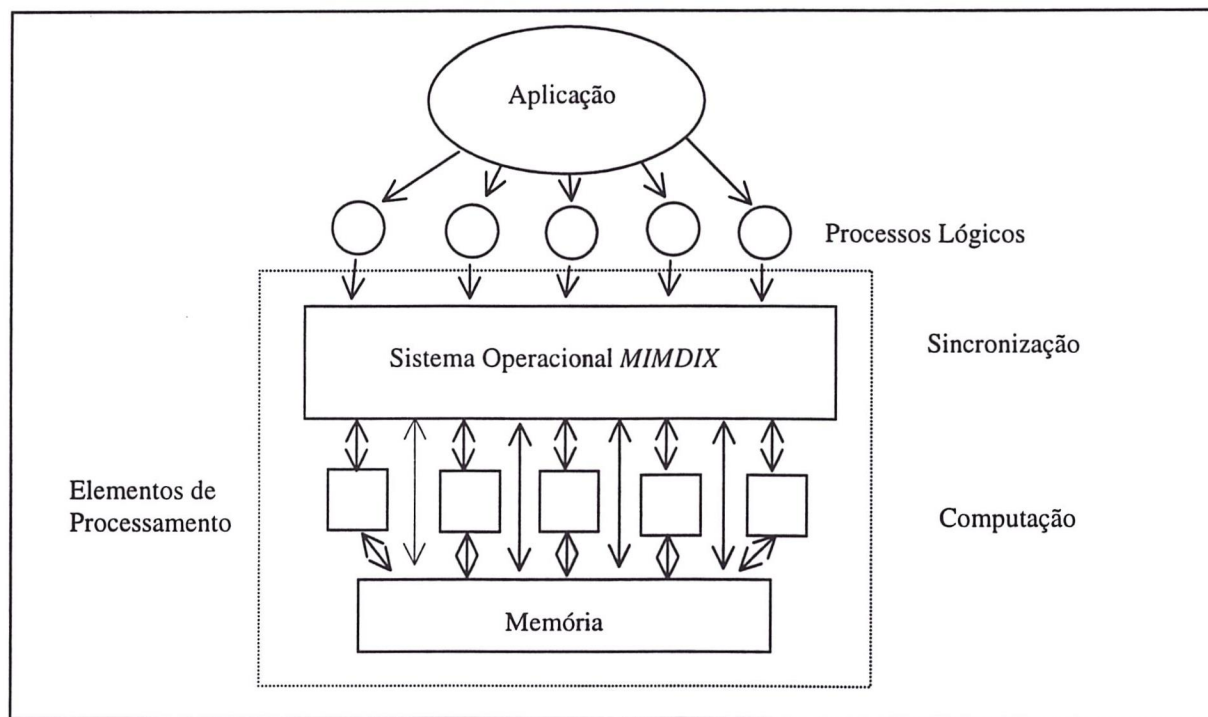


Figura 4.3: Ambiente de Simulação Paralela *MIMDIX*.

O mecanismo de sincronização é totalmente transparente ao usuário/programa de aplicação, e a memória pode ser compartilhada ou distribuída entre os processadores.

Em intervalos de tempo δ determinados probabilisticamente, os *LPs* são sincronizados por um processo especial, que inicia e manipula a sincronização probabilística, e executa em um processador dedicado. Esse processo atualiza o *GVT* do sistema e envia periodicamente um *SYNC broadcast* (com marca de tempo um pouco maior do que *GVT*) para o sistema, impedindo que os *LPs* avancem muito no tempo simulado [Ferscha 1995a, Rajaei et al. 1993].

A re-sincronização é executada em cada *LP* após o recebimento da mensagem *SYNC*, os quais descartam todas as mensagens com marcas de tempo maior ou igual que a marca de tempo da mesma. A coleta de lixo é executada todas as vezes que o *kernel* determina que não existe memória disponível para um *LP* armazenar a mensagem que chega.

É interessante ressaltar que o sistema ainda envia antimensagens (como no *Time Warp*) sempre que uma computação errônea acontece. Outro aspecto interessante do sistema refere-se ao fato de que a aplicação pode escolher o valor de δ e as probabilidades para sincronização (que dependem da aplicação e do projeto de memória).

Vantagens [Madisetti et al. 1992]:

- Foram observados bons *speedups* com relação ao ambientes anteriores (*Time Warp*);
- A utilização de memória é considerada excelente, não sendo necessária nenhuma técnica sofisticada de gerenciamento de lista de eventos;
- O sistema tem controle sobre o progresso da simulação, podendo otimizar o uso da memória e/ou *speedup*;

- O usuário/programa de aplicação não sabe da existência do mecanismo de sincronização;
- O mecanismo de sincronização pode adaptar-se à natureza da computação, tanto para otimizar o uso de memória quanto para aumentar a velocidade da computação.

Desvantagens:

- A aplicação pode escolher o valor de δ e as probabilidades para sincronização;
- Se o intervalo de re-sincronização é pequeno, o mecanismo se torna muito conservativo. Se é grande, a re-sincronização deixa de ter sentido [Rajaei et al. 1993];
- *LPs* podem desfazer trabalho correto;
- Apagar mensagens depois de cada SYNC pode se tornar uma fonte de *overhead*, embora pareça reduzir a limitação de memória do *Time Warp* [Rajaei et al. 1993];
- A necessidade do conhecimento probabilístico sobre o comportamento do sistema sendo simulado;
- A existência de um elemento centralizado para cuidar da sincronização.

4.14 Algoritmos Adaptativos *NPSI*

Os protocolos adaptativos *Near Perfect State Information (NPSI)* são protocolos otimistas nos quais a agressividade e o risco são controlados dinamicamente através de informação de estado quase perfeitas [Srinivasan; Reynolds 1995a, Srinivasan; Reynolds 1995b].

4.15 *Unified Distributed Simulation (UDS)*

Desenvolvido em 1990, fornece um mecanismo de janelas deslizantes para envio e recebimento de mensagens. As mensagens com marcas de tempo fora dos limites das janelas não são enviadas ou recebidas, mas os *LPs* são bloqueados, o que pode levar a um *deadlock* [Rajaei et al. 1993].

4.16 *WOLF*

Desenvolvido em 1988. Quando ocorre *rollback*, são enviadas mensagens especiais em forma de *broadcast*, para encerrar rapidamente a computação errônea [Rajaei et al. 1993]. Tais mensagens impedem o avanço da computação incorreta em processos remotos devido ao fato de que são enviadas diretamente, ao invés da forma indireta e recursiva com que são enviadas as antimensagens [Das 1996].

Desvantagem:

- Pode impedir o progresso de computações corretas, bloqueando-as desnecessariamente, visto que o mecanismo não sabe distinguir quais mensagens de eventos são corretas e quais não são. Isso também vale para o mecanismo *Moving Time Windows* (seção 4.9) [Prakash; Subramanian 1991].

4.17 Window-based Throttling

Nesse método, um *LP* somente pode executar as mensagens de eventos cujas marcas de tempo estão dentro de uma janela de tempo [Reynolds 1989].

Desvantagens:

- Cada aplicação pode necessitar de uma janela de tamanho diferente, e esse valor pode mudar até mesmo de *LP* para *LP*;
- Pode impedir *LPs* de executar, mesmo quando não irão executar mensagens de eventos que irão provocar *rollback*;
- Tamanho da janela muito pequeno degrada o desempenho, e muito grande perde o sentido.

4.18 Penalty-based Throttling

Outro método de limitar o otimismo no *Time Warp* é manter controle sobre quais *LPs* estão fazendo trabalho correto e quais estão fazendo trabalho que deverá ser descartado. Os primeiros têm mais ciclos de *CPU* para trabalhar, e os outros, menos ciclos. Assumindo que esse comportamento tende a persistir em um futuro próximo, diminuem-se as chances de *rollback* e portanto pode-se melhorar o desempenho [Reynolds 1989].

Vantagem:

- Melhor que *Window-based Throttling*.

Desvantagem:

- Penalidades pequenas apresentaram pouco ou nenhum efeito;
- Penalidades grandes não melhoraram o resultado de forma significativa;
- O melhor desempenho obtido foi um *speedup* de 7%.

4.19 Adaptive Bounded Time Window

Proposto em 1993, é similar ao *Moving Time Window*. A largura da janela é controlada dinamicamente, baseado no conceito de trabalho útil executado por um *LP*. Trabalho útil corresponde a uma medida do trabalho produtivo executado por um *LP*, mostrada em função de uma série de parâmetros (como a razão do número de eventos

executados corretamente e o número de eventos em relação ao número total de eventos executados, número de *rollbacks*, tamanho médio do *rollback*, número de antimensagens enviadas, etc). Cada *LP* possui seu próprio valor da janela, que aumenta ou diminui periodicamente, dependendo da mudança do trabalho útil executado [Das 1996].

4.20 Protocolo Probabilístico Baseado em Custos

Proposto em 1995, é similar ao *Adaptive Time Warp*, porém mais sofisticado. O método está baseado em uma função de custos probabilística, utiliza um modelo de custos envolvendo a probabilidade e o custo de *rollbacks*. Cada *LP* monitora a marca de tempo e o instante real da chegada de cada mensagem, assim como o comportamento de *rollback*. O modelo de custos é utilizado para avaliar a largura ótima para uma *blocking window* [Das 1996].

4.21 Local Adaptive Protocol (LAP)

Protocolo desenvolvido em 1994. Tenta estimar uma *blocking window* de tempo real, baseado em incrementos médios do *LVT* e tempos médios entre-chegadas (tempo de *CPU* e tempo simulado) observados nos canais de entrada. Utiliza mensagens nulas para prevenir *deadlocks* [Ferscha 1995a].

Um *LP* bloqueia se estima que um evento chegando tarde em um de seus canais de entrada vazios irá causar *rollback* [Das 1996].

Usa informações estatísticas para determinar se é melhor executar um evento mesmo correndo o risco de *rollback*, ou atrasar sua execução, para evitar *rollbacks* e antimensagens [Ferscha 1995b].

Desvantagem:

- Uso de mensagens nulas, que pode levar a *overhead* de comunicação.

4.22 Composite ELSA

Algoritmo que unifica os conceitos de controle otimista e conservativo. O sistema simulado é modelado com o um grafo dirigido onde os vértices correspondem aos *LPs* e os arcos a interconexão entre os nós [Arvind; Smart 1992].

Um nó pode chavear entre modo conservativo ou modo otimista quando determina que um evento é seguro ou não [Das 1996]. Podem ocorrer *rollbacks* locais e globais e não há a noção de *GVT* [Rajaei et al. 1993].

Vantagem:

- Unifica as abordagens otimistas e conservativas, agrupando as vantagens de ambas.

Desvantagem:

- O desempenho cai conforme aumenta o número de processadores [Rajaei et al. 1993].

4.23 *Conservative-Optimistic Protocol (CO-OP)*

Apresentado em 1991, esse método faz com que um *LP* não selecione sempre a primeira mensagem da lista de eventos. Se alguma condição não é satisfeita o *LP* é bloqueado temporariamente e fica esperando por mensagens menos propensas a erros [Rajaei et al. 1993]. Ou seja, esse protocolo processa os eventos em termos de sua probabilidade de estar correto, não estritamente por ordem de tempo (menor marca de tempo primeiro).

Para entender melhor, pode-se imaginar uma situação onde o *LP* com o primeiro evento a ser processado sabe que deve esperar por outra mensagem antes de manipular aquele evento (por exemplo, uma resposta a uma consulta). Processando eventos usando esse conhecimento de probabilidade de correteude, pode-se eliminar muitos *rollbacks* [Steinman 1992].

Desvantagem:

- Como verificar a probabilidade de correteude.

4.24 *Filtered Rollback Algorithm*

Desenvolvido em 1989, combina *Bounded Lag* com *Time Warp*. Segundo Das [Das 1996], podem ocorrer violações de causa e efeito, e são utilizados *rollbacks* e cancelamento de eventos como no *Time Warp*. Dessa forma, o protocolo é agressivo e com risco.

4.25 *Tentative Time Warp (TTW)*

Fornece uma novo mecanismo de controle local, baseado em cancelamento de grupo, para simulação otimista. O mecanismo de controle global permanece como no *Time Warp* [Kalantery 1997].

Vantagens [Kalantery 1997]:

- Introduz o *Tentative Event Cancellation*, ao invés de utilizar Cancelamento Agressivo ou Cancelamento Preguiçoso;
- Reduz o número de mensagens para cancelamento;
- A busca na lista de eventos para propagação de *rollbacks* primários é eliminada, e minimizada para *rollbacks* secundários;
- Fornece uma nova estrutura de ordenação dos eventos, que facilita o cancelamento.

4.26 *Adaptive Throttle Scheme*

Baseado no fato de que o desempenho do *Time Warp* depende muito do progresso de cada *LP* no tempo, propõe o uso da *Global Progress Window (GPW)*, para indicar como progridem no tempo o *LP* mais lento e o mais rápido. Esse esquema ajusta a velocidade de execução dos eventos no *Time Warp*, permitindo aos *LPs* acelerar ou suspender sua execução.

Como os *LPs* podem entrar em um “estado estacionário”, as recuperações excessivas de *rollbacks* podem ser evitadas.

Para quantificar o o efeito de *rollbacks* e para o propósito de comparar diferentes implementações *Time Warp*, propõe duas novas medidas denominadas *RPE* (número de eventos que levam a *rollbacks* versus eventos corretamente executados) e ξ (efetividade em reduzir o *overhead* causado pelos *rollbacks*) [Tay et al. 1997].

4.27 ADAPT

Tenta preencher a lacuna entre os métodos otimistas e conservativos. Permite que diferentes partes de um modelo de simulação sejam executadas utilizando protocolos otimistas ou conservativos. O modo de execução para um submodelo pode ser alterado dinamicamente [Jha; Bagrodia 1994].

4.28 Length-based Blocking

Estratégia de bloqueamento de *LPs* que o autor classifica como preventiva porque previne os *LPs* antes da ocorrência de um erro de causa e efeito. Um *LP* permanece bloqueado até que o comprimento de sua fila de entrada ultrapasse um valor limite. Como motivação, ressalta que em muitos casos a probabilidade de encontrar uma violação de causa e efeito diminui conforme o número de eventos na fila aumenta (o *LP* pode escolher um evento para executar). Consequentemente, o bloqueamento é uma maneira de diminuir a probabilidade de *rollbacks* permitindo que mais eventos sejam inseridos na na fila de entrada antes de qualquer execução [Pham; Fdida 1996].

Vantagem:

- O mecanismo apresenta vantagem quando o tempo gasto com o bloqueio não é maior do que o tempo gasto com *rollbacks* [Pham; Fdida 1996].

5. Tabelas Comparativas

O grande número de protocolos que procuram limitar o comportamento otimista do *Time Warp* torna difícil a comparação entre eles. As tabelas a seguir procuram resumir suas principais características, o que pode facilitar a compreensão. A tabela 5.1 apresenta todos os mecanismos estudados, o tipo de método utilizado na sincronização, os mecanismos com os quais se relaciona e a classificação (Das ou Snirivasan) a qual pertence. As demais tabelas dividem os mecanismos em grandes grupos, baseados em janelas (tabela 5.2), probabilísticos (tabela 5.3), grupo (tabela 5.4), conhecimento (tabela 5.5) e estado (tabela 5.6).

Mecanismo	Tipo de Sincronização	Relacionado com	Classificação
<i>Lock Step Time Driven</i>	Executa os eventos em intervalos regulares com incremento de tempo fixo		
<i>Wolf</i> (1988)	Quando ocorre <i>rollback</i> , envia <i>broadcast</i> para encerrar rapidamente a computação errônea	<i>Time Warp</i>	Das, Srinivasan
<i>Filtered Rollback Algorithm</i> (1989)	Adiciona otimismo a protocolo de janelas conservativo	combina <i>Bounded Lag</i> e <i>Time Warp</i>	

<i>Penalty-based Throttling</i> (1989)	Penaliza os <i>LPs</i> que efetuam muito <i>rollback</i>	<i>Time Warp</i>	Das, Srinivasan (ambas)
<i>Window-based Throttling</i> (1989)	Janelas	<i>Time Warp</i>	Srinivasan
<i>Moving Time Windows</i> (1989)	Janelas	<i>Time Warp</i>	Das, Srinivasan
<i>UDS</i> (1990)	Janelas deslizantes	<i>Time Warp</i>	Srinivasan (AAWP)
<i>Adaptive Time Warp</i> (1990)	Janelas (<i>blocking window</i>) adaptativas	<i>Time Warp</i>	Das, Srinivasan (AAWP)
<i>SRADS</i> (1990)	usa a idéia de simulação especulativa porém as filas de eventos e os estados podem ser alterados, não usa antimensagens	Simulação Especulativa	Srinivasan
<i>Time Buckets</i> (1991)	Ciclos determinados pelo horizonte de eventos	<i>Lock Step Time Driven</i>	Nenhuma
<i>Breathing Time Buckets</i> (1991)	Ciclos determinados pelo horizonte de eventos	<i>Time Buckets e Time Warp</i>	Nenhuma
<i>Filter</i> (1991)	Cada mensagem carrega uma lista de supostas classes de mensagens que poderiam levar ao seu cancelamento, e os <i>LPs</i> trocam informações	<i>Time Warp</i>	Das, Srinivasan
<i>Simulação Especulativa</i> (1991)	Executa os eventos em <i>LPs</i> que estão desocupados, sem alterar o estado e a fila de eventos; faz análise dos dados para verificar a correteude	<i>Time Warp</i>	Nenhuma
<i>Conservative-Optimistic Protocol (CO-OP)</i> (1991)	Processa os eventos em termos de sua probabilidade de estarem corretos, não estritamente por ordem de tempo	<i>Time Warp</i> , pois utiliza <i>rollbacks</i>	Das
<i>SFilter</i> (1992)	Cada mensagem carrega suposições locais	<i>Filter</i>	Nenhuma
<i>Composite ELSA</i> (1992)	O sistema simulado é visto como um grafo dirigido; cada nó pode chavear entre otimista e conservativo	otimista e conservativo	Das
<i>Breathing Time Warp</i> (1992)	Horizonte de eventos e processamento otimista	<i>Time Warp, Breathing Time Buckets</i>	Das, Srinivasan (ambas)
<i>Bounded Time Warp</i> (1992)	Janelas	<i>Time Warp</i>	Das, Srinivasan
<i>MIMDIX</i> (1992)	Sincroniza os <i>LPs</i> em intervalos δ determinados probabilisticamente	<i>Time Warp</i> (utiliza antimensagens)	Das, Srinivasan
<i>Local Time Warp</i> (1993)	Sincronização otimista dentro dos grupos de <i>LPs</i> , e conservativa entre os grupos	<i>Time Warp e Conservative Time Windows</i>	Das, Srinivasan
<i>Adaptive Bounded Time Window</i> (1993)	Cada <i>LP</i> possui seu próprio valor da janela, que aumenta ou diminui periodicamente, dependendo da mudança do trabalho útil executado	<i>Time Warp</i>	Das
<i>Aggressive Windowing</i> (1993)	Janelas (conservativa e agressiva), antimensagens, cancelamento agressivo e <i>rollback</i>	<i>Time Warp</i>	Nenhuma
<i>Local Adaptive Protocol</i> (1994)	Tenta estimar uma <i>blocking window</i> de tempo real	<i>CMB</i> , usa mensagens nulas para prevenir <i>deadlock</i>	Das, Srinivasan (AAWP)
<i>Probabilistic DDES Protocol</i> (1994)	Utiliza janelas cujo tamanho é variável e calculado probabilisticamente	<i>Time Warp</i>	Das
<i>PADOC</i> (1995)	Bloqueia temporariamente o processamento dos eventos, de forma probabilística	<i>CMB e Time Warp</i>	Das

Algoritmos Adaptativos <i>NPSI</i> (1995)	Agressividade e o risco são controlados dinamicamente através de informação de estado quase perfeitas		Das, Srinivasan (AAWP)
Protocolo Probabilístico Baseado em Custos (1995)	Calcula probabilisticamente uma largura ótima para uma <i>blocking window</i>	ATW	Das
<i>Length-based Blocking</i> (1996)	Bloqueia <i>LPs</i> até que a fila de entrada atinja um valor limite	<i>Time Warp</i>	Nenhuma
<i>Adaptive Throttle Scheme</i> (1997)	Ajusta a velocidade de execução dos <i>LPs</i>	<i>Time Warp</i>	Nenhuma
<i>Tentative Time Warp</i> (1997)	Mecanismo de controle local baseado em cancelamento de grupos; o controle global permanece como no <i>Time Warp</i>	<i>Time Warp</i>	Nenhuma

Tabela 5.1: Resumo das Características dos Mecanismos que Limitam Otimismo no *Time Warp*.

Mecanismo	Análise
<i>Window-based Throttling</i>	os <i>LPs</i> fora da janela são bloqueados
<i>Moving Time Windows</i>	as janelas tem tamanho pré-definido
<i>UDS</i>	os <i>LPs</i> são bloqueados por períodos de janelas
<i>Adaptive Time Warp</i>	bloqueia os <i>LPs</i> por um período denominado <i>blocking window</i> (janelas adaptativas)
<i>Time Buckets</i>	as janelas são determinadas pelo horizonte de eventos
<i>Breathing Time Warp</i>	os primeiros <i>N</i> eventos são processados como no <i>Time Warp</i> , depois utiliza <i>Breathing Time Buckets</i>
<i>Bounded Time Warp</i>	para determinar quando todos os <i>LPs</i> estão inativos, utiliza um <i>token</i> que circula entre eles
<i>Adaptive Bounded Time Window</i>	as janelas são calculadas dinamicamente, de acordo com o trabalho útil executado pelos <i>LPs</i>
<i>Adaptive Throttle Scheme</i>	ajusta a velocidade de execução dos eventos no <i>Time Warp</i> , permitindo aos <i>LPs</i> acelerar ou suspender sua execução
<i>Aggressive Windowing</i>	

Tabela 5.2: Comparação dos mecanismos de janelas.

Mecanismo	Análise
<i>Conservative-Optimistic Protocol (CO-OP)</i>	Esse protocolo processa os eventos em termos de sua probabilidade de estar correto, não estritamente por ordem de tempo (menor marca de tempo primeiro).
<i>MIMDIX</i>	Em intervalos determinados probabilisticamente, os <i>LPs</i> são sincronizados por um processo especial
<i>Probabilistic Distributed Discrete Event Simulation Protocol</i>	Utiliza janelas com tamanho variável, em função das mensagens que chegam em cada <i>LP</i>
<i>Protocolo Probabilístico Baseado em Custos</i>	Utiliza uma função probabilística envolvendo a probabilidade e o custo de <i>rollbacks</i> para determinar uma <i>blocking window</i> , ou seja, um período para o <i>LP</i> ficar bloqueado
<i>PADOC</i>	Cada <i>LP</i> monitora as mensagens que chegam e pelo padrão de chegada conseguem se adaptar à próxima mensagem, segundo a abordagem otimista ou conservativa

Tabela 5.3: Comparação dos mecanismos probabilísticos.

Mecanismo	Análise
<i>Local Time Warp</i>	Utiliza <i>Time Warp</i> dentro dos grupos e um protocolo conservativo entre os grupos
<i>Tentative Time Warp</i>	Fornece uma novo mecanismo de controle local, baseado em cancelamento de grupo, para simulação otimista. O mecanismo de controle global permanece como no <i>Time Warp</i>

Tabela 5.4: Comparação dos mecanismos de grupo.

Mecanismo	Análise
<i>Filter</i>	Cada mensagem carrega uma lista de suposições sobre o que pode causar <i>rollback</i>
<i>Sfilter</i>	Como <i>Filter</i> , porém não necessita guardar as mensagens enviadas
<i>WOLF</i>	Faz broadcast para encerrar a computação errônea

Tabela 5.5: Comparação dos mecanismos baseados em conhecimento.

Mecanismo	Análise
Algoritmos Adaptativos <i>NPSI</i>	Utilizam informações baseado no estado local para controlar o otimismo
<i>LAP</i>	Tenta estimar uma blocking window de tempo real, baseado em incrementos médios do LVT e tempos médios entre-chegadas. Envolve os conceitos de janelas e estudos probabilísticos.
<i>Penalty-based Throttling</i>	Penaliza os <i>LPs</i> que executam mais <i>rollbacks</i>
<i>Filtered Rollback</i>	Utiliza uma estimativa da distância mínima no tempo de simulação entre um par de <i>LPs</i> (denominada <i>lag</i>) como base para determinar quais são os eventos seguros para processar
<i>Length-based Blocking</i>	Um <i>LP</i> permanece bloqueado até que sua fila de entrada atinja um tamanho limite
<i>Composite ELSA</i>	Um <i>LP</i> pode chavear entre modo conservativo ou modo otimista quando determina que um evento é seguro ou não

Tabela 5.6: Comparação dos mecanismos baseados em informações de estado.

6. Linguagens e Ambientes para Simulação Distribuída

Segundo Bagrodia [Bagrodia 1994], a área de linguagens e ambientes de simulação paralelos ainda está no início de seu desenvolvimento. Muitos estudos devem ser feitos para o projeto de ferramentas paralelas de propósito geral, sua aplicação em domínios específicos, e na identificação de características da aplicação que possam ser exploradas por protocolos de sincronização específicos. Ressalta que os ambientes de simulação paralelos disponíveis fornecem um suporte mínimo para balanceamento de carga estático e a maioria não suporta balanceamento dinâmico ou migração de processos.

As linguagens de simulação paralelas diferem no modo como suportam os protocolos de sincronização. Muitas suportam somente protocolos otimistas (*Sim++*, *ModsimII*, *Apostle*, *SPEEDES*). Poucas suportam somente protocolos conservativos, e algumas suportam ambos. Apesar do fato de que os protocolos conservativos serem mais de implementar, os protocolos otimistas são mais utilizados.

A seguir são apresentados alguns ambientes e linguagens para simulação distribuída.

ParSMPL

ParSMPL é um ambiente de simulação paralela que utiliza o mecanismo *CMB* para efetuar a sincronização entre os processos lógicos. É uma extensão da extensão funcional sequencial *SMPL*, e utiliza como plataforma o ambiente de passagem de mensagens *PVM*. [Ulson et al. 1997].

ModSim II

É uma linguagem de simulação orientada a objetos baseada na linguagem *Modula 2*. Utiliza estrutura de blocos, tipagem forte, encapsulamento e herança múltipla. Os componentes primários de um programa *ModSim* são os objetos, composto de variáveis de estado (campos) e funções (métodos). Fornece construções para criação de objetos dinâmicos, comunicação e sincronização. Utiliza o sistema operacional *TWOS* para sincronizar a execução de programas paralelos *ModSimII* [Bagrodia 1994].

GPSS Paralelo

Knop [Knop et al. 1996] apresenta uma paralelização da linguagem de simulação *GPSS*. Foi implementado um tradutor *GPSS-C++* e o código transformado é executado com o auxílio do ambiente de simulação paralela *ParaSol*. Segundo o autor, o mapeamento de *GPSS* para C++ é simples porque *ParaSol*, diferente dos demais ambientes de simulação distribuída, é orientado a transação. Ainda segundo os autores, o fato de *GPSS* ser uma linguagem sequencial leva a algumas construções com desempenho fraco quando se utiliza paralelismo.

Parasol

Parasol [Knop 1996, Mascarenhas et al. 1996] é um sistema de simulação distribuída projetado para suportar transações, abordagem utilizada por muitas linguagens sequenciais como *CSIM*, *GPSS* e *SIMAN*. Ao invés de utilizar mensagens com marcas de tempo para comunicação e sincronização entre os *LPs*, utiliza transações que migram de forma transparente entre os *LPs*. Foi desenvolvido em C++ e *PVM*.

Extensão de *CSim* para Simulação Distribuída

Utiliza a ferramenta existente (a linguagem *Csim*) para definir submodelos que irão executar em processadores diferentes, e extensões da ferramenta definindo as interfaces entre eles. As extensões englobam toda a comunicação e sincronização necessárias.

SPEEDES (Synchronous Parallel Environment for Emulation and Discrete Event Simulation)

SPEEDES é um ambiente de simulação baseado em C++ que implementa um conjunto de protocolos de sincronização incluindo *Time Warp* e um algoritmo de janelas denominado *Breathing Time Buckets* [Bagrodia 1994, Steinman 1991].

Maise

É uma linguagem de simulação baseada em C que pode utilizar tanto simulação sequencial quanto simulação distribuída (mecanismos otimistas e conservativos). É também a primeira linguagem que suporta otimização semântica (uso de semânticas para reduzir o *overhead* no uso das abordagens otimistas e conservativas) [Bagrodia 1994].

Portable Parallel Discrete Event Simulation System

O sistema foi projetado para facilitar a simulação de aplicações reais utilizando protocolos PDES, na *University of Exeter*. O sistema fornece meios para comparar protocolos de simulação, podendo ser utilizado para verificar a adequabilidade das abordagens otimistas ou conservativas em áreas de aplicação específicas. Também permite converter código de simulação sequencial de forma que este possa ser executado em um computador paralelo, evitando-se assim o custo de reescrevê-lo. Outra característica deste sistema é permitir o projeto e a avaliação de novos protocolos para simulação paralela e para investigação de variações de protocolos existentes. Uma abordagem interessante envolve o desenvolvimento de protocolos híbridos, pela adição de características conservativas às abordagens otimistas e vice-versa [Turner; Xu 1992b].

Tudo que o usuário necessita conhecer é o processamento de eventos na camada de aplicação, sem se preocupar com os detalhes do protocolo de simulação que está sendo utilizado.

Como resultado do desenvolvimento do sistema, foi criado um protocolo híbrido denominado *Bounded Time Warp (BTW)*, resultado da adição de características conservativas ao protocolo otimista *Time Warp*.

COMPOSE (Conservative, Optimistic and Mixed Parallel Object-oriented Simulation Environment)

COMPOSE suporta a execução de um modelo utilizando um algoritmo de simulação adaptativo onde os submodelos podem ser sincronizados utilizando métodos conservativos ou otimistas, e um *LP* pode mudar seu modo de sincronização de forma dinâmica [Martin; Bagrodia 1996].

PORTS (Parallel, Optimistic, Real-Time Simulator)

PORTS é um sistema de simulação que executa em ambientes que impõem regras de tempo real na execução do programa de simulação [Ghosh et al. 1994].

PARSEC (Parallel Simulation Environment for Complex Systems)

O ambiente de simulação fornece interfaces com o usuário como as linguagens *Parsec* e *Compose* e um editor gráfico para construção do modelo de simulação. Implementa vários protocolos de sincronização (otimistas e conservativos) utilizando uma biblioteca de comunicação portátil, o que o torna disponível para uma variedade de plataformas de

hardware (*IBM SP, Sun Sparc 1000*, máquinas monoprocessadoras utilizando *Unix, Windows NT* ou *Windows 95*, redes de estações de trabalho) [Bagrodia 1998].

7. Conclusão

Um sistema de simulação é tipicamente executado como uma repetição dos seguintes passos sequenciais: buscar um evento em uma lista, executar um passo da simulação e, possivelmente, atualizar a lista e o relógio da simulação. A natureza do mecanismo de lista de eventos impõe uma característica sequencial, desde que em cada ciclo da simulação somente um item é removido da lista de eventos, seus efeitos são simulados, e possivelmente a lista de eventos é atualizada. Esse algoritmo não pode ser prontamente adaptado para execução concorrente em vários processadores, uma vez que a lista de eventos não pode ser dividida entre os processadores [Misra 1986]. Além disso, a simulação sequencial demanda altos requisitos computacionais, conforme o tamanho do problema aumenta. Com os avanços na tecnologia, mais e mais pesquisadores estão encontrando maneiras de executar a simulação em paralelo. A pesquisa na comunidade de simulação paralela apresenta primariamente o objetivo de desenvolver tecnologias e ferramentas para possibilitar que programas de simulação sejam desenvolvidos e executados no menor tempo possível [Fujimoto 1995].

As pesquisas que vêm sendo desenvolvidas em simulação paralela têm, na sua maioria, procurado resolver o problema da sincronização, seja com novas propostas, seja com variações dos protocolos *CMB* e *Time Warp*. Outros tópicos que agora vêm merecendo destaque são gerenciamento de memória, balanceamento de carga, ferramentas de desenvolvimento. Existem pontos importantes que tornam o desenvolvimento de software para simulação paralela mais difícil do que simulação sequencial, como o particionamento do modelo em processos lógicos.

Sem uma escolha clara entre os mecanismos otimistas e conservativos, muita atenção tem sido depositada no desenvolvimento de mecanismos que mesclam as duas abordagens, seja adicionando agressividade a protocolos conservativos, seja procurando limitar o excesso de otimismo nos protocolos otimistas. Segundo Rajaei [Rajaei; Ayani 1993], a principal desvantagem de limitar o otimismo consiste em como encontrar um limite adequado para melhorar o desempenho de forma significativa.

Foram descritos vários protocolos que procuram reduzir o excesso de riscos no mecanismo *Time Warp*, com janelas que limitam o avanço dos *LPs* no tempo, probabilidades que impedem um *LP* de prosseguir caso estejam ocorrendo muitos *rollbacks*, ou então ainda mecanismos que dividem os *LPs* em grupos e aplicam um protocolo entre os *LPs* do grupo, e outro protocolo entre os grupos.

8. Referências Bibliográficas

- AYANI, Rassul; RAJAEI, Hassan. Parallel Simulation using Conservative Time Windows. *Proceedings of the 1992 Winter Simulation Conference*, p. 709-717, 1992.
- AYANI, Rassul; RAJAEI, Hassan. Parallel Simulation Based on Conservative Time Windows: a Performance Study. *Concurrency: Practice and Experience*, v. 6, n. 2, pp. 119-142, April 1994.
- ARVIND, D. K.; SMART, C. R.. Hierarchical Parallel Discrete Event Simulation in Composite ELSA. *Proceedings of the 6th Workshop on Parallel and Distributed*

- Simulation (PADS'92)*, p. 147-156, 1992.
- BAGRODIA, Rajive L.. Language Support for Parallel Discrete-Event Simulations. *Proceedings of the 1994 Winter Simulation Conference - WSC'94*, p. 1324-1331, 1994.
- BAGRODIA, Rajive L.. Perils and Pitfalls of Parallel Discrete-Event Simulation. *Proceedings of the 1996 Winter Simulation Conference (WSC'96)*, p. 136-146, 1996.
- BAGRODIA, Rajive L.; MEYER, Richard; TAKAI, Mineo; CHEN, Yu-an; ZENG, Xiang; MARTIN, Jay; SONG, Ha Yoon. PARSEC: A Parallel Simulation Environment for Complex Systems. *IEEE Computer*, v. 3, n. 10, p. 77-85, October 1998.
- BALL, Duane; HOYT, Susan. The Adaptive Time Warp Concurrency Control Algorithm. *Simulation Series*, v. 22, n. 11, p. 174-177. The Society for Computer Simulation, 1990.
- CHANDY, K. M.; MISRA, J.. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, p. 440-452, v. SE-5, September 1979.
- CHIOLA, G.; FERSCHA, A.. Performance Comparable Implementation Design of Synchronization Protocols for Distributed Simulation. *Technical Report Series of the Austrian Center for Parallel Computation*, ACPC/TR 93-23; December 1993.
- DAS, Samir R.. Adaptive Protocols for Parallel Discrete Event Simulations. *Proceedings of the 1996 Winter Simulation Conference*, p. 186-193, 1996.
- DJEMAME, Karin; BETTAZ, Mohamed; GILLES, Dennis C.; MACKENZIE, Lewis M.. Performance Comparison of High Level Algebraic Nets Distributed Simulation Protocols. *Proceedings of the 1996 Winter Simulation Conference (WSC'96)*, p. 621-628, 1996.
- FERSCHA, Alois; CHIOLA, G.. Self-Adaptive Logical Processes: the Probabilistic Distributed Simulation Protocol. *Proceedings of the 27th Annual Simulation Symposium*, p. 78-88. IEEE Computer Society Press, April 1994.
- FERSCHA, Alois. Probabilistic Adaptive Direct Optimism Control in Time Warp. *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS'95)*, p. 120-129, IEEE Computer Society Press, 1995.
- FERSCHA, Alois. Adaptive Model Parallelism Exploitation in Parallel Simulation. *Proceedings of EUROSIM'95*, p. 241, 248, September 1995.
- FERSCHA, Alois. Parallel and Distributed Simulation of Discrete Event Systems. In *Parallel and Distributed Computing Handbook*. Chapter 35. McGraw-Hill, 1995.
- FERSCHA, Alois. Estimating Rollback Overhead for Optimism Control in Time Warp. *Proceedings of the 28th Annual Simulation Symposium*, p 2-12, 1995.
- FUJIMOTO, Richard M.. Parallel Discrete Event Simulation. *Communications of the ACM*, v. 33, n. 10, p. 31-53, tober 1990.
- FUJIMOTO, Richard M.. Parallel and Distributed Discrete Event Simulation: Algorithms and Applications. *Proceedings of the 1993 Winter Simulation Conference*, 1993.
- FUJIMOTO, Richard M.. Parallel and Distributed Simulation. *Proceedings of the 1995 Winter Simulation Conference*, 1995.
- GHOSH, Kaushik; PANESAR, Kiran; FUJIMOTO, Richard M.; SCHWAN, Karsten. PORTS: A Parallel, Optimistic, Real-Time Simulator. *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS'94)*, 1994.

- HOEGER, Herbert R.; JONES, Douglas W.. Integrating Concurrent and Conservative Distributed Discrete-Event Simulations. *Simulation*, v. 67, n. 5, p. 303-314, Novembro 1996.
- JHA, Vikas; BAGRODIA, Rajive L.. A Unified Framework for Conservative and Optimistic Distributed Simulation. *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS'94)*, p. 12-19, 1994.
- JEFFERSON, David R.. Virtual Time. *IEEE Transactions on Programming Languages and Systems*, 7(3):404-425, July 1985.
- KALANTERY, Nasser. Tentative Time Warp. *Proceedings of the 3th International Euro-Par Conference. In Parallel and Distributed Algorithms, Lecture Notes in Computer Science*, v. 1300, p. 458-467, 1997.
- KNOP, Felipe; MASCARENHAS, Edward; REGO, Vernon. A Parallel GPSS Based on the Parasol Simulation System. *Proceedings of the 1996 Winter Simulation Conference (WSC'96)*, p. 801-808, 1996.
- MADISSETTI, Vikay K.; HARDAKER, David A, FUJIMOTO, Richard M.. The MIMDIX Operating System for Parallel Simulation. *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS'92)*, p. 65-64, 1992.
- MARTIN, Jay M.; BAGRODIA, Rajive L.. COMPOSE: An Object-Oriented Environment for Parallel Discrete-Event Simulations. *Proceedings of the 1995 Winter Simulation Conference (WSC'95)*.
- MASCARENHAS, Edward; KNOP, Felipe; REGO, Vernon. Parasol: A Multithreaded System for Parallel Simulation based on Mobile threads. *Proceedings of The 1995 Winter Simulation Conference*, 1995.
- MISRA, Jayadev. Distributed Discrete-Event Simulation. *ACM Computing Surveys*, v. 18, n. 1, p. 39-65, 1986.
- PHAM, C. D.; FDIDA, S.. Length-based Blocking and Local Estimations in Distributed Simulation: A Case Study. *Proceedings of the 29th Symposium on Simulation*, p.97-106, April 1996.
- PHAM, C. D.; BAGRODIA, R. L.. Building Parallel Time-Constrained HLA Federates: A case Study with the PARSEC Parallel Simulation Language. *Proceedings of the 1998 Winter Simulation Conference (WSC'98)*, December 1998.
- PRAKASH, Atul; SUBRAMANIAN, Rajalakshmi. Filter: An Algorithm for Reducing Cascaded Rollbacks in Optimistic Distributed Simulations. *Proceedings of the 24th Annual Simulation Symposium, Simulation Multiconference*, IEEE Press, pp. 123-132, April 1991.
- PRAKASH, Atul; SUBRAMANIAN, Rajalakshmi. An Efficient Optimistic Distributed Simulation Scheme based on Conditional Knowledge. *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS'92)*. Simulation Series, v. 24, n. 3, 1992
- RADHAKRISHNAN, Radharamanan; MOORE, Lantz; WILSEY, Philip A.. External Adjustment of Runtime Parameters in Time Warp Synchronized Parallel Simulators. *Proceedings of the 11th International Parallel Processing Symposium*, p.260-266, April 1997.
- RADHAKRISHNAN, Radharamanan; ABU-GHAZALEH, Nael; CHETLUR, Malolan; WILSEY, Philip A.. On-line Configuration on a Time Warp Parallel Discrete Event Simulator. *Proceedings of the International Conference on Parallel Processing (ICPP-98)*, p. 28-35, August 1998.

- RAJAEI, Hassan; AYANI, Rassul; THORELLI, Lars-Erik. The Local Time Warp Approach to Parallel Simulations. *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS'93)*, p. 119-126, 1993.
- REIHER, Peter L.; JEFFERSON, David. Limitation of Optimism in the Time Warp Operating System. *Proceedings of the 1989 Winter Simulation Conference (WSC'89)*, . 765-769. 1989.
- REIHER, Peter L.; FUJIMOTO, Richard; BELLENOT, Steven; JEFFERSON, David. Cancellation Strategies in Optimistic Execution Systems. *Proceedings of the 1990 Distributed Simulation Conference*, p. 112-121, 1990.
- REYNOLDS Jr., Paul F.. Comparative Analysis of Parallel Simulation Protocols. *Proceedings of the 1989 Winter Simulation Conference (WSC'89)*, p. 671-679, 1989.
- RÖNNGREN, Robert; LILJENSTAM, Michael; AYANI, Rassul; MONTAGNAT, Johan. A Comparative Study of State Saving Mechanisms for Time Warp Synchronized Parallel Discrete Event Simulation. *Proceedings of the 24th Annual Simulation Symposium (ASS'96)*, March 1996.
- SKÖLD, Sven; RÖNNGREN, Robert. Event Sensitive State Saving in Time Warp Parallel Discrete Event Simulations. *Proceedings of the 1996 Winter Simulation Conference (WSC'96)*; December 1996.
- SOKOL, Lisa M.; STUCKY, Brian K.. MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm. *Simulation Series*, v. 22, n. 11, p. 1169-173, The Society for Computer Simulation, 1990.
- SPOLON, Renata; SANTANA, Marcos J.; SANTANA, Regina H. C.. Distributed Simulation, Time Warp and its Variants: Taxonomy and Performance Evaluation Issues. *Proceedings of the 13th European Simulation Multiconference (ESM'99)*, p. 220-227, June 1999.
- SRINIVASAN, Sudhir; REYNOLDS Jr.; Paul F.. NPSI Adaptive Synchronization Algorithms for PDES. *Proceedings of the 1995 Winter Simulation Conference (WSC'95)*, p. 658-665, 1995.
- SRINIVASAN, Sudhir; REYNOLDS Jr.; Paul F.. Adaptive Algorithms vs. Time Warp: An Analytical Comparison. . *Proceedings of the 1995 Winter Simulation Conference (WSC'95)*, p. 666-673, 1995.
- STEINMAN, Jeffrey S.. SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation. *Proceedings of SCS Multiconference on Advances on Parallel and Distributed Simulation*, p. 95-103, 1991.
- STEINMAN, Jeffrey S.. SPEEDES: A Unified Approach to Parallel Simulation. . *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS'92)*, p. 75-84, 1992.
- STEINMAN, Jeffrey S.. Breathing Time Warp. *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS'93)*, p. 109-118, 1993.
- STEINMAN, Jeffrey S.. Scalable Parallel and Distributed Military Simulations using the Speedes Framework. *Electronic Conference on Scalability in Training Simulation*. 1995 (www.mystech.com/~smithr/elecsim95/PAPERS/steinman/steinman-main.html)
- TAY, Seng C.; TEO, Yong M.; KONG, Siew T.. Speculative Parallel Simulation with an Adaptive Throttle Scheme. *Proceedings of the 11th Workshop on Parallel and Distributed Simulation (PADS'97)*, p. 116-123, June 1997.
- TURNER, Stephen J.; XU, Ming Q.. Performance Evaluation of the Bounded Time Warp

Algorithm. *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS'92)*, p. 117-126, 1992.

TURNER, Stephen J.; XU, Ming Q.. A Portable Parallel Discrete Event Simulation System. *Proceedings of ESPRIT Workshop on Parallel Computing*, p. 284-287. March 1992.

ULSON, Roberta S.; SANTANA, Regina H. C.; SANTANA, Marcos J.. Distributed Simulation Environment for Computing Systems Performance Evaluation. *The Proceedings of the 1997 Summer Computer Simulation Conference - Symposium of Performance Evaluation of Computer and Telecommunication Systems (SPECTS 97)*, pp. 85-90, The Society for Computer Simulation International. July 1997. ISBN 1-56555-123-0.

VRIES, Ronald C.. Reducing Null Messages in Misra's Distributed Discrete Event Simulation Method. *IEEE Transactions on Software Engineering*, v. 16, n. 1, p. 82-91, January 1990.

WILSEY, Philip A.. Feedback Control in Time Warp Synchronized Parallel Simulators. *First International Workshop on Distributed Interactive Simulation and Real Time Applications*, p.31-38, January 1997.