Universidade de São Paulo
Instituto de Física de São Carlos
Departamento de Física e Informática

Um Método para Avaliação de Desempenho de Protocolos de Sincronização Otimistas para Simulação Distribuída

Renata Spolon

São Carlos
2001

Universidade de São Paulo
Instituto de Física de São Carlos
Departamento de Física e Informática

Um Método para Avaliação de Desempenho de Protocolos de Sincronização Otimistas para Simulação Distribuída

Renata Spolon

Tese apresentada ao Instituto de Física de São Carlos, da Universidade de São Paulo, para a obtenção do título de Doutor em Ciências (Física Aplicada – opção em Física Computacional).

Orientador: Prof. Dr. Marcos José Santana

São Carlos
2001

Spolon, Renata
Um Método para Avaliação de Desempenho de Protocolos de Sincronização Otimistas para Simulação Distribuída/ Renata Spolon. São Carlos, 2001.
p. 247

Tese (Doutorado) – Instituto de Física de São Carlos, 2001.

Orientador: Prof. Dr. Marcos José Santana

1. Avaliação de Desempenho 2. Simulação Distribuída.
3. Protocolos Otimistas.
I. Título.

A sabedoria não nos é dada; é preciso descobri-la por nós mesmos depois de uma viagem que ninguém nos pode poupar ou fazer por nós.
Marcel Proust

Aos meus pais Dorival e Meiri

À minha irmã Roberta

Ao Daniel

Pelo amor, carinho e incentivo, que tornaram possível esta conquista.

Agradecimentos

Ao Prof. Dr. Marcos José Santana, pela incansável e valiosa orientação e, principalmente, pelo apoio, amizade e confiança, sem os quais este trabalho não teria sido realizado.

Aos meus pais, que representam a razão e o incentivo deste trabalho, pela compreensão durante os muitos dias de ausência e pelo seu esforço incondicional investido na formação dos filhos.

À minha irmã Roberta, pelo apoio e companheirismo fundamentais em todos os momentos e pelas críticas e sugestões indispensáveis para este trabalho.

Ao Daniel, pelo amor e carinho que me ajudam a enfrentar os obstáculos da vida, e pela compreensão nos momentos em que estive ausente.

À Profa. Dra. Regina Helena Carlucci Santana, pelas contribuições para a realização deste trabalho.

Ao Marcos e à Regina, pela confiança em mim depositada desde as disciplinas da graduação, passando pela iniciação científica, pelo mestrado e pelo doutorado. Pela amizade, pelos conselhos, pela dupla orientação e por serem a minha família em São Carlos.

Ao amigo João Carlos de Moraes Morselli Jr. (Boca), pela troca de idéias e pelas palavras encorajadoras durante todo o doutorado.

Ao Prof. Dr. Eduardo Marques, do ICMC-USP, e ao Prof. Dr. Nivaldi Calônego Jr., da UNIC, pelos ensinamentos e pelo permanente incentivo.

À Wladerez, da seção de pós-graduação do IFSC-USP, pela simpatia e eficiência com que sempre me atendeu.

Ao Márcio Augusto de Souza e à Sarita Mazzini Bruschi, sempre dispostos a ajudar, pela amizade e pelas muitas dúvidas solucionadas.

Aos colegas do Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP, Álvaro, Andrezza, Boca, Célia, Edmilson, João Carlos, Jorge, Kalinka, Luciano, Mara, Márcio Augusto, Mário, Omar, Paulo Sérgio e Simone, Renato, Ricardo, Sarita, Tatiana, Tomás e Vera, por tornarem o LaSDPC um ótimo ambiente de trabalho.

Ao pessoal do STI-ICMC-USP, pela atenção: Eduardo, Cabral, Franz e Sônia.

Ao Prof. Dr. Edson Norberto Cáceres, do DCT/CCET/UFMS, pelo apoio.

Ao Departamento de Computação e Estatística da UFMS, pelo meu afastamento. À Giselda e ao Sr. Leodir, por sua atenção na secretaria do DCT.

À Leonora e à Daisy, da Pró-Reitoria de Pesquisa e Pós-Graduação da UFMS, pela eficiência ao cuidarem dos assuntos relacionados ao meu afastamento.

To Bradd Righby (*Alphatech, Inc.*), for helping with *ALPHA/Sim*.

Àqueles que direta ou indiretamente contribuíram para a realização deste trabalho.

Este trabalho teve auxílio financeiro da Capes e da UFMS.

Sumário

Capítulo 6

Avaliação dos Protocolos *Time Warp* Básico e *PDSP*

Capítulo 7

Conclusões, Contribuições e Propostas para Trabalhos Futuros

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AAWP	*Asynchronous Adaptive Waiting Protocols*
ATW	*Adaptive Time Warp Concurrency Control Algorithm*
BW	*Blocking Window*
BTB	*Breathing Time Buckets*
BTW	*Breathing Time Warp*
Compose	*Conservative Optimistic and Mixed Parallel-oriented Simulation Environment*
OFC	*Optimistic Fossil Collection*
CO-OP	*Conservative-Optimistic Protocol*
DIS	*Distributed Interactive Simulation*
GSPN	*Generalized Stochastic Petri Nets*
GPW	*Global Progress Window*
GVT	*Global Virtual Time*
LAP	*Local Adaptive Protocol*
LVT	*Local Virtual Time*
LTW	*Local Time Warp*
LVTW	*Local Virtual Time Windows*
MTW	*Moving Time Windows*
NPSI	*Near Perfect State Information*
PADOC	*Probabilistic Direct Optimism Control*
PDSP	*Probabilistic Distributed discrete event Simulation Protocol*
ROSS	*Rensselaer's Optimistic Simulation System*
SFilter	*Single-hop Filter*
SPEEDES	*Synchronous Parallel Environment for Emulation and Discrete Event Simulation*
SPN	*Stochastic Petri Nets*
Estelle	*Extended State Transition Language*
TTW	*Tentative Time Warp*
UDS	*Unified Distributed Simulation*

Resumo

Esta tese propõe um novo método para a avaliação de desempenho de protocolos de sincronização otimistas adotados em programas de simulação distribuída. A utilização do método proposto é atrativa principalmente por permitir um usuário de simulação distribuída otimista optar pelo protocolo que mais se adapta às suas necessidades, sem ter que implementá-lo. O método de avaliação tem o modelo de um processo lógico *Time Warp* básico como núcleo, levando em consideração as características tanto da aplicação como da plataforma de hardware/software utilizada (o ambiente *ALPHA*/*Sim* é utilizado para a modelagem e simulação através de Redes de Petri). As características mais relevantes do protocolo *Time Warp* básico e suas variações são identificadas, a partir de uma nova taxonomia hierárquica proposta nesta tese, aliada a uma revisão bibliográfica ampla e detalhada. Além disso, domínios de *plugins* foram identificados, permitindo uma extensão (ou modificação) rápida e direta do modelo do *Time Warp* básico proposto. Esta tese também contribui apresentando resultados da avaliação de desempenho do protocolo *Time Warp* básico e do protocolo probabilístico *PDSP*.

Abstract

This thesis proposes a novel method for the performance evaluation of optimistic synchronisation protocols adopted in distributed simulation programs. The use of the proposed method is attractive mainly for allowing an optimistic distributed simulation user to choose the protocol that best fits his needs, without having to implement it. The evaluation method has the model for a basic Time Warp logical process as its kernel, that takes into consideration the features of both the application and the hardware/software platform adopted (the ALPHA/Sim environment is used for modelling and simulation purposes through Petri Nets). The more relevant features of the basic Time Warp protocol and its variants are identified by using a new hierarchical taxonomy proposed in this thesis together with a wide and detailed literature survey. Furthermore, plugin domains could also been identified, allowing a quick and straightforward extension (or modification) of the basic Time Warp model proposed. This thesis also contributes by presenting performance evaluation results for both the basic Time Warp protocol and the PDSP probabilistic protocol.

Capítulo 1

Introdução

Este capítulo discute a motivação e os objetivos desta tese e a organização do trabalho.

1.1 Motivação

O processamento paralelo e distribuído tem se popularizado nos últimos anos, motivado pela possibilidade de aumento do desempenho na execução de diferentes tipos de aplicações, quando comparadas com o desempenho obtido através da computação seqüencial.

Uma das áreas da computação que tem se tornado essencial com essa grande evolução dos sistemas computacionais é a avaliação de desempenho, uma vez que, indicando a qualidade do trabalho executado por um sistema, pode ser utilizada na seleção de sistemas alternativos, no projeto de novos sistemas e na análise comparativa de sistemas existentes. Para esse fim existem as técnicas de aferição (*benchmarks*, coleta de dados e protótipos) e as técnicas de modelagem (com solução analítica e com solução por simulação). A escolha de qual ferramenta utilizar depende, fundamentalmente, do objetivo do estudo em andamento e do tipo de avaliação desejada.

A complexidade dos sistemas computacionais resulta em modelos também mais complexos e a solução através da simulação seqüencial pode tornar o tempo de

execução inviável para o estudo de desempenho. Uma solução para diminuir esse tempo é dividir a simulação em diversos processos e executá-los em paralelo, adotando assim a simulação distribuída (Fujimoto 1990a).

A união dos conceitos de simulação e computação paralela e distribuída possibilitou o desenvolvimento das técnicas de simulação distribuída que surgem como alternativa para o estudo de aplicações complexas, cujas soluções dos modelos demandam grande potência computacional.

A estrutura da simulação tradicional, inerentemente seqüencial devido à existência da lista de eventos futuros, teve que sofrer adaptações para incorporar os conceitos de computação distribuída. Novos mecanismos foram desenvolvidos para garantir a execução correta dos eventos da simulação, isto é, em ordem não decrescente do seu tempo de ocorrência, como na simulação seqüencial. Os conceitos de sincronização de processos da computação distribuída levaram ao desenvolvimento de protocolos, classificados como conservativos ou otimistas, para garantir a sincronização entre os processos da simulação distribuída, evitando assim (ou corrigindo) os chamados erros de causa e efeito (Fujimoto 2000).

Em um protocolo de sincronização conservativo os tempos de ocorrência dos eventos são verificados para que haja a certeza de que nenhum evento seja executado fora da ordem cronológica (Misra 1986). Desse modo o protocolo garante que os eventos da simulação são executados na ordem imposta pelo sistema que está sendo simulado.

Nos protocolos otimistas, dos quais o principal representante é o *Time Warp*, os eventos são executados sem que haja qualquer verificação quanto à sua ordem cronológica (Carothers et al. 2000, Jefferson 1985). Quando um evento fora de ordem é detectado, o mecanismo de sincronização é responsável por desfazer toda a computação executada de forma errônea, através de *rollbacks*, e por restaurar o estado da simulação para um estado consistente. A partir do estado a que foi retrocedida, a simulação continua a execução dos eventos.

Como a resposta à questão sobre a superioridade dos protocolos de sincronização otimistas em relação aos conservativos (ou vice-versa) é complexa e a construção de regras para tal fim é uma tarefa bastante difícil, a alternativa tem sido o desenvolvimento de protocolos variantes que procuram limitar o excesso de otimismo dos protocolos otimistas ou inserir otimismo nos protocolos conservativos

(Das 1996, Srinivasan; Reynolds 1995a, Srinivasan; Reynolds 1995b). O primeiro caso é o objeto de estudo deste trabalho.

1.2 Objetivos

Existe uma grande variedade de protocolos de sincronização que buscam limitar o excesso de otimismo da proposta original (Jefferson 1985), o que torna difícil a opção do usuário da simulação distribuída por essa ou aquela implementação. A comparação entre eles é difícil e também confusa, uma vez que as taxonomias ou classificações descritas na literatura não conseguem agrupar, de maneira clara, os protocolos existentes. Além disso, falham por não englobarem protocolos mais recentes e não levarem em consideração as características da implementação, que podem influenciar um estudo de desempenho entre protocolos de sincronização. O grande número de variantes do *Time Warp* torna difícil o trabalho de escolha de um determinado protocolo otimista, quando se deseja utilizar a simulação distribuída para resolver uma aplicação. Para realizar essa escolha, pode-se efetuar uma comparação de desempenho entre os protocolos e utilizar aquele com melhor resultado. A avaliação de desempenho de protocolos de sincronização em simulação distribuída constitui uma tarefa complexa, pois deve considerar não apenas as diferentes estratégias de implementação e otimização, como também as características das plataformas de hardware e ambientes de software utilizados e da aplicação que está sendo resolvida pela simulação distribuída.

O estudo de desempenho de um protocolo de sincronização de simulação distribuída pode ser efetuado utilizando-se as mesmas técnicas de aferição e modelagem utilizadas para verificar o desenvolvimento de sistemas computacionais (Francês et al. 1997). Essas técnicas apresentam vantagens e desvantagens, sendo que só é possível aferir um sistema existente. A modelagem é aplicada quando o sistema em estudo ainda não foi implementado, ou quando aferir o sistema real não é possível ou não é desejável. Já a resolução analítica de um modelo apresenta resultados mais precisos, porém a dificuldade é proporcional à complexidade do modelo. Dessa forma, a representação do protocolo de sincronização em um modelo e a solução desse modelo através de simulação torna-se atrativa quando se deseja inserir modificações e avaliar os efeitos dessas modificações.

Do ponto de vista do usuário da simulação distribuída, existe uma carência de ferramentas práticas de avaliação dos diversos protocolos que vêm sendo apresentados na literatura, uma vez que grande número dos trabalhos utiliza casos específicos de teste e necessita de um ambiente real (a implementação do protocolo) para que se possam realizar as análises (Chiola; Ferscha 1995, Fabbri; Donatiello 1997, Jha; Bagrodia 1996, Porras et al. 1998).

Com o objetivo de facilitar o trabalho do usuário da simulação distribuída otimista, esta tese propõe um método que permite a avaliação de desempenho dos protocolos de sincronização em simulação distribuída otimista. Por se adaptar mais às necessidades do trabalho, a solução de modelo através de simulação é a abordagem adotada. Com isso, o usuário da simulação distribuída pode obter resultados de desempenho de um protocolo de sincronização otimista, sem a necessidade de implementá-lo. O estudo de desempenho de protocolos de simulação distribuída através da construção e experimentação de modelos exigiu que uma nova taxonomia fosse proposta, identificando e agrupando protocolos semelhantes, assim como as principais características da sincronização com base no paradigma de tempo virtual do *Time Warp* (Jefferson 1985). Isso justifica o fato do capítulo 3 apresentar parte de revisão bibliográfica sobre os protocolos que limitam o otimismo e as classificações existentes. Esse estudo serviu de base para a nova taxonomia proposta que procura minimizar o impacto causado pelas falhas das taxonomias apresentadas na literatura.

O objetivo do trabalho não é apresentar a avaliação de desempenho de um protocolo específico, mas apresentar o método para desenvolver essa avaliação. Ou seja, assume-se que o usuário já optou pelo paradigma da simulação distribuída e por utilizar protocolos otimistas, e tem conhecimento das características dos protocolos e de como utilizar a simulação distribuída. Com isso, pode-se fornecer flexibilidade ao usuário que pode optar pelo protocolo que melhor se adapta às suas necessidades.

O método proposto nesta tese é inovador sob diversos aspectos, destacando-se por permitir uma fácil caracterização de diferentes arquiteturas e protocolos otimistas a serem avaliados, sem a necessidade de implementação, fornecendo dessa forma uma previsão do comportamento da simulação distribuída.

1.3 Organização

O capítulo 2 desta tese apresenta uma revisão sobre os principais conceitos de simulação distribuída e o protocolo otimista *Time Warp*, objeto de estudo deste trabalho. As Redes de Petri, que serão utilizadas no capítulo 4, são discutidas. São abordados os diferentes tipos de Redes de Petri, suas classificações e como utilizá-las para efetuar a avaliação de desempenho de sistemas.

No capítulo 3 faz-se uma revisão sobre os principais protocolos de sincronização variantes do *Time Warp* e as taxonomias existentes na literatura. Essas taxonomias são analisadas criticamente, o que permite apontar suas falhas e a necessidade de melhorias. A partir dessa revisão e da análise crítica é proposta uma nova taxonomia, mais abrangente e hierárquica, com o objetivo de facilitar o estudo da variedade de protocolos otimistas. A nova taxonomia não considera apenas as diferentes formas de se inserir variações e limitações no *Time Warp*, mas também diversas opções de algoritmos que podem ser utilizados em sua implementação. Esses algoritmos não descaracterizam um determinado protocolo, porém são importantes em seu desempenho e por isso foram considerados no desenvolvimento da taxonomia proposta. Nesta nova taxonomia, o *Time Warp* de Jefferson (Jefferson 1985) é considerado como *Time Warp* básico e os demais protocolos otimistas são protocolos variantes do básico.

No capítulo 4 é apresentada a descrição do comportamento de um processo lógico que utiliza *Time Warp* para sincronização da simulação distribuída. Essa descrição serve como alicerce para a definição de um novo modelo básico de um processo lógico, que é verificado e validado. A especificação do modelo é feita através das Redes de Petri. É apresentada a definição de domínios de *plugins* (definidos a partir da taxonomia proposta no capítulo 3), que permitem estender e ou modificar o comportamento do modelo básico proposto, para representar protocolos variantes do *Time Warp*.

O capítulo 5 apresenta um método definido para avaliação de desempenho do *Time Warp* básico e dos protocolos variantes, definidos a partir da taxonomia apresentada no capítulo 3. No método, cada processo lógico é visto como uma caixa preta, através de blocos básicos que permitem a representação da simulação distribuída sem a necessidade de trabalhar diretamente com a representação em

Redes de Petri. É discutido como conectar esses blocos básicos através de exemplos e a introdução da representação da plataforma de execução da simulação distribuída (através de *plugins*).

No capítulo 6 é apresentado um estudo do protocolo *Time Warp* básico e do protocolo *PDSP*, exemplificando como o método proposto pode ser utilizado na avaliação de desempenho de protocolos de sincronização em simulação distribuída otimista.

O capítulo 7 encerra o trabalho, apresentando as contribuições resultantes, as conclusões obtidas e os planos para a continuidade do trabalho.

Capítulo 2

Revisão Bibliográfica

Este capítulo faz uma revisão sobre os principais conceitos que envolvem a realização de uma simulação distribuída, apresentando o problema de sincronização e as técnicas utilizadas para resolvê-lo. O protocolo otimista *Time Warp*, objeto de estudo deste trabalho, é detalhado. São apresentadas também as características das Redes de Petri, suas classificações e como o processo de modelagem utilizando essa técnica pode ser aplicado à representação de sistemas e sua evolução no tempo.

2.1. Considerações Iniciais

Uma simulação seqüencial de um sistema complexo pode exigir grande esforço computacional e um tempo de processamento que pode inviabilizar o estudo. Esse cenário tem sido alterado devido ao uso de ambientes onde a computação paralela pode ser explorada, como no caso dos supercomputadores e de ambientes constituídos por vários elementos de processamento comunicando-se através de uma rede. A simulação distribuída foi proposta por Chandy e, independentemente, por Bryant em 1977 (Bryant 1977, Chandy; Misra 1979) e vem se desenvolvendo muito nos últimos vinte anos. Como exemplo de aplicações que estão se beneficiando com o uso de simulação distribuída, podem-se citar a simulação de redes de telecomunicações e redes de longa distância, circuitos digitais, sistemas

computacionais (Fujimoto 2000, Jones; Das 2000, Simmonds et al. 2000), sistemas de produção (Zhang et al. 2000), etc.

A principal questão relacionada com a simulação distribuída refere-se à necessidade de sincronização entre os processos que a constituem, e para isso utilizam-se dois tipos básicos de protocolos, os otimistas e os conservativos, a partir dos quais muitas variantes têm sido propostas.

O estudo de protocolos de sincronização pode ser efetuado de diferentes formas. Uma técnica adequada para a representação de protocolos é o desenvolvimento de modelos, que neste trabalho foi realizado através de Redes de Petri. As Redes de Petri têm como característica o fato de permitirem a representação de atividades concorrentes e um alto grau de detalhamento nas características do sistema em estudo (Jensen 1998) e, por isso, foram utilizadas. No capítulo 4 são apresentadas as técnicas para avaliação de desempenho de protocolos para simulação distribuída e justifica-se a escolha pelas técnicas de modelagem e Redes de Petri, que fornecem a base para a modelagem desenvolvida neste trabalho.

Neste capítulo são descritos os fundamentos da simulação distribuída (seção 2.2), as principais características do protocolo conservativo *CMB* (seção 2.3) e o protocolo otimista *Time Warp*, objeto de estudo deste trabalho (seção 2.4). O capítulo apresenta também uma revisão sobre a aplicação das Redes de Petri (seção 2.5) como técnica de representação e simulação de modelos. A seção 2.6 apresenta as considerações finais do capítulo.

2.2 Fundamentos de Simulação Distribuída

A simulação distribuída refere-se à execução de um único programa de simulação em um sistema computacional paralelo ou distribuído. Esse assunto tem sido muito investigado nos últimos vinte 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 seqüenciais. Embora o objetivo principal da simulação distribuída seja a redução no tempo de processamento, pode-se tirar proveito da tolerância a falhas e da distribuição geográfica de um sistema distribuído e também do uso de sistemas arquiteturais diferentes em uma arquitetura paralela virtual. Dessa forma, a

simulação distribuída vem sendo aplicada em diferentes áreas, como por exemplo no estudo de redes de telecomunicações e de longa distância, circuitos digitais, sistemas computacionais e jogos de guerra (Alleyne; Tropper 2000, Fujimoto 1990a, Fujimoto 2000, Kelly et al. 2000, McGough; Mitrani 2000, Simmonds et al. 2000, Unger et al. 2000).

A razão da dificuldade de se fazer simulação distribuída torna-se evidente quando se analisa a operação de uma simulação seqüencial. A simulação seqüencial geralmente utiliza três estruturas de dados:

- 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 remove o evento com a menor marca de tempo da lista de eventos, avança o relógio da simulação para o tempo de ocorrência desse evento e executa as ações necessárias para a ocorrência do mesmo. Ao terminar, o programa de simulação retira o próximo evento da lista e repete os passos descritos.

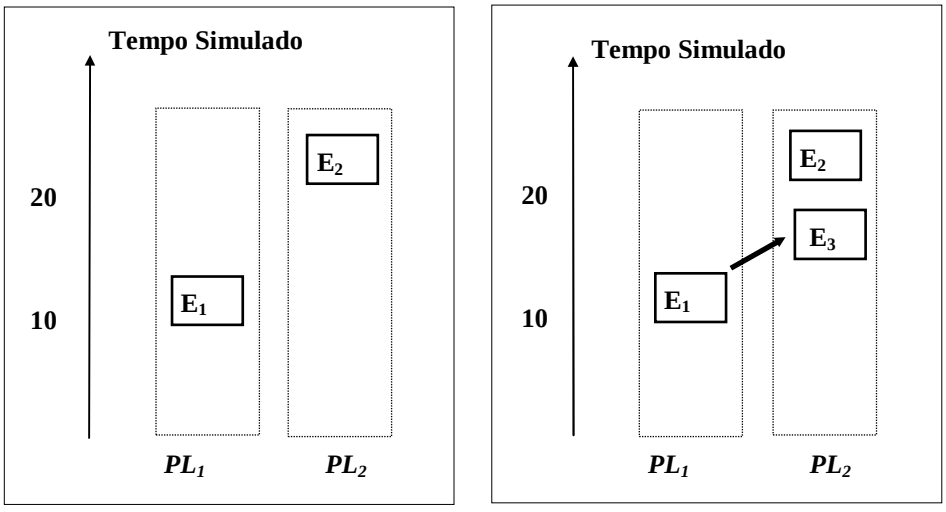
Nesse paradigma de execução é crucial que sempre seja selecionado o evento com a menor marca de tempo (E_{min}) como o próximo evento a ser executado. Isso porque se outro evento for selecionado pode haver alteração no valor das variáveis de estado utilizadas por E_{min}. Ou seja, estaria-se simulando um sistema no qual o futuro poderia influenciar o passado. Essa situação é denominada erro de causa e efeito (Fujimoto 2000, Misra 1986).

Considerando a paralelização de um programa de simulação com base no paradigma descrito anteriormente, verifica-se que a grande oportunidade para paralelismo surge do processamento de eventos concorrentes em diferentes processadores. Entretanto, um mapeamento direto desse 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 1990a, Fujimoto 1993).

No paradigma da simulação distribuída 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 PL_0, PL_1,..., onde cada processo lógico representa um processo do sistema físico (o sistema real). 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 correspondente ao processo físico que modela, além de um relógio local, e um conjunto de variáveis de estado que não são compartilhadas entre os mesmos (Fujimoto 2000).

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 PL_1 (marca de tempo 10) e E_2 no PL_2 (marca de tempo 20), como mostrado na figura 2.1(a). Se E_1 escalona um novo evento E_3 para o processo PL_2 contendo uma marca de tempo menor do que 20, E_3 poderia afetar E_2, o que implica na necessidade de execução seqüencial dos três eventos (figura 2.1(b)).



(a) (b)

Figura 2.1: Ocorrência de Erros de Causa e Efeito.

Para preservar a ordem de causa e efeito entre os eventos, são necessários protocolos que sincronizem os tempos de simulação dos processos lógicos. Tais protocolos são classificados em duas categorias: conservativos e otimistas (alguns autores propõem uma terceira categoria, a dos mecanismos mistos, com submodelos que executam em modo otimista ou conservativo (Bagrodia et al. 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 processos lógicos podem processar um evento somente quando as relações de causa e efeito são preservadas. Nesse caso, podem ocorrer *deadlocks* (Misra 1986).

As abordagens otimistas utilizam algoritmos que permitem o progresso da simulação até onde for possível e, então, quando uma violação de causa e efeito ocorrer, utilizam um mecanismo de *rollback* para recuperar os estados dos processos lógicos, (Bagrodia 1996, Ferscha 1995c, Fujimoto 2000, Hoeger; Jones 1996, Jha; Bagrodia 1994, Knop et al. 1996, Pham; Bagrodia 1998, Radhakrishnan et al. 1998).

2.3 Protocolos Conservativos e o Protocolo *CMB*

Os primeiros protocolos de simulação distribuída foram baseados em abordagens conservativas. O problema básico que esses protocolos devem resolver é determinar quando é seguro executar um evento. Isto é, se um processo lógico contém um evento não executado E_1 com marca de tempo T_1 (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_1, então o processo pode executar E_1 seguramente porque pode garantir que fazendo isso não irá resultar em uma violação de causa e efeito. Os processos lógicos que contêm eventos que não são seguros devem ser bloqueados, o que pode levar a situações de *deadlocks* se não forem tomadas precauções adequadas (Misra 1986).

Os primeiros algoritmos para simulação paralela desenvolvidos de maneira independente por Chandy e Misra (Chandy; Misra 1979) e Bryant (Bryant 1977) (por isso a denominação *CMB*) 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, exige-se que a seqüência 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.

Cada canal possui um relógio associado que é igual:

- A marca de tempo da mensagem do início da fila, se a fila contém uma mensagem;
- A marca de tempo da última mensagem recebida, se a fila está vazia;
- Zero se nenhuma mensagem tiver sido recebida.

O relógio do processo lógico (*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 2.2).

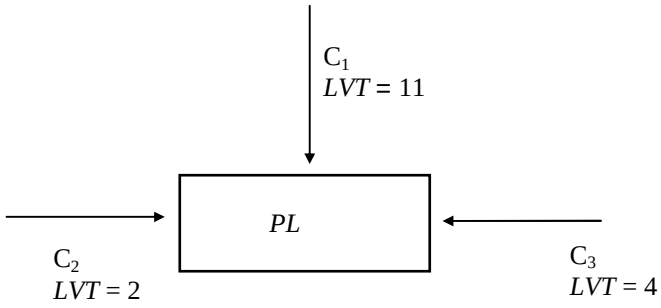


Figura 2.2: *LVT* de um Processo Lógico.

O processo lógico seleciona repetidamente o canal com o menor *LVT* 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 *LVT*, já que elas irão chegar em ordem cronológica ao longo da cada canal. Se a fila selecionada está vazia, o processo lógico entra em estado de bloqueio. 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 processo lógico é forçado a esperar por uma mensagem para atualizar o relógio no canal antes que possa atualizar seu *LVT* (Fujimoto 1990a, Misra 1986).

Um problema crítico nesse protocolo ocorre quando cada processo lógico 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 processos lógicos bloqueados e cada um está bloqueado devido a outro processo no ciclo. Como exemplo, pode-se considerar a situação da figura 2.3 (Fujimoto 1990a). Cada processo lógico está esperando no canal de entrada contendo o menor valor de relógio porque a fila correspondente está vazia. Todos os três estão bloqueados, embora existam mensagens em outras filas.

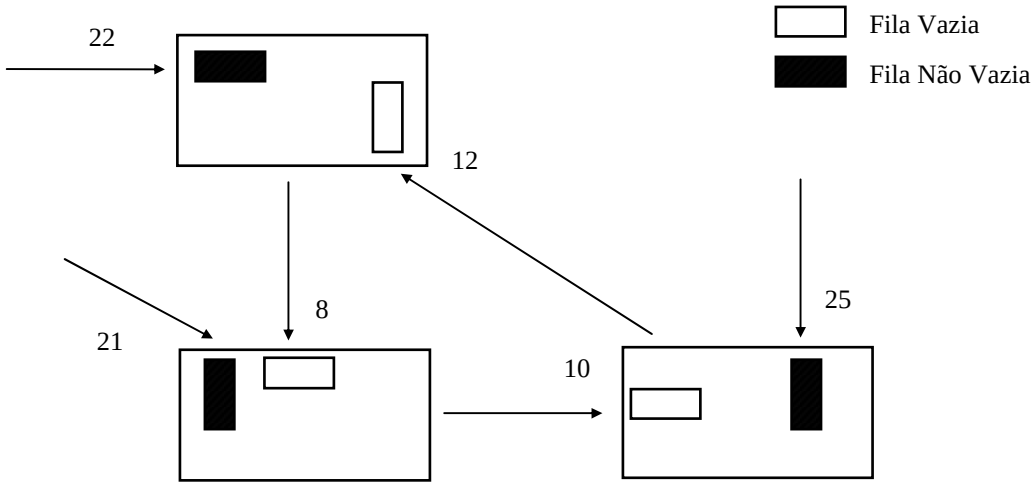


Figura 2.3: Situação de *Deadlock*.

Para evitar que a simulação entre em *deadlock*, foi desenvolvido um protocolo que utiliza mensagens nulas quando não há mensagens reais a serem transmitidas (Misra 1986). As mensagens nulas servem somente para sincronização, e não correspondem a qualquer atividade no sistemas físico. São tratadas como qualquer outra mensagem quando recebidas por um processo lógico, isto é, a recepção de uma mensagem nula faz com que o processo atualize seu *LVT*. Assim, uma mensagem nula com marca de tempo *T*, enviada por um PL_i para um PL_j indica que PL_i não irá enviar mensagens a PL_j entre o valor corrente do relógio e o tempo *T*. Por essa razão, qualquer mensagem futura de PL_i para PL_j terá marca de tempo

superior a *T*. Um exemplo de implementação do protocolo *CMB* com mensagens nulas é o ParSMPL (Ulson et al. 1997)

Uma desvantagem do protocolo *CMB* consiste na sobrecarga provocada pelas mensagens nulas. Uma variação da abordagem de mensagens nulas consiste em enviar mensagens nulas sob demanda e não após o processamento de cada evento. A freqüência dessa demanda pode ser dada por um *timeout* ou quando o menor relógio de todos os canais for o de uma fila vazia, indicando que o processo lógico está bloqueado. Quando isso ocorre, a próxima mensagem é requisitada (pode ser uma mensagem nula ou não) do processo que envia para esse canal. O processo lógico continua sua execução quando a resposta a esse pedido for recebida.

Segundo Fujimoto (Fujimoto 1990a) a maior desvantagem dos mecanismos conservativos consiste no fato de que estes não podem explorar totalmente o paralelismo disponível na aplicação. Se for 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. No pior caso a abordagem conservativa é pessimista, e força a execução seqüencial quando não é necessária.

2.4 Protocolos Otimistas e o Protocolo *Time Warp*

Os protocolos otimistas detectam e recuperam erros de causa e efeito, ao invés de evitá-los. Em contraste com os protocolos conservativos, as estratégias otimistas não necessitam determinar quando é seguro executar um evento, pois quando um erro é detectado, a simulação é restaurada para um estado consistente. Uma vantagem dessa abordagem em relação aos protocolos conservativos consiste no fato de que a mesma permite explorar o paralelismo em situações onde erros poderiam ocorrer, mas de fato não ocorrem (Fujimoto 2000).

O protocolo otimista *Time Warp* é objeto de estudo deste trabalho. Exemplos de implementações do protocolo *Time Warp* incluem *Warped* (Radhakrishnan et al. 1998) e *GTW* (Fujimoto 2000).

O *Time Warp* básico é o protocolo de simulação distribuída otimista mais difundido (Chetlur et al. 1998, Djemame et al. 1996, Pasquini; Rego 1999, Fujimoto

2000) e tem como base o paradigma de tempo virtual proposto por Jefferson (Jefferson 1985). Nesta tese os demais protocolos otimistas são considerador variantes do *Time Warp* básico e são abordados no capítulo 3.

O *Time Warp* possui dois componentes principais, o mecanismo de controle local, que garante que os eventos sejam executados na ordem correta, e o mecanismo de controle global, responsável pelo gerenciamento de memória e detecção do fim da simulação. Esses mecanismos são descritos nas próximas seções.

Um processo lógico não necessita enviar as mensagens em ordem cronológica (marca de tempo), e o meio de comunicação não garante que as mensagens sejam recebidas na mesma ordem em que foram enviadas. Cada processo lógico é constituído pelas seguintes estruturas de dados (figura 2.4) (Jefferson 1985):

- Relógio local: *LVT*;
- Lista de entrada de mensagens (*Input Queue*): armazena as mensagens recebidas pelo processo. Cada mensagem na *Input Queue* armazena as informações referentes à marca de tempo, o processo lógico transmissor, o processo lógico receptor, o sinal (no *Time Warp*, uma mensagem possui sinal positivo e uma antimensagem possui sinal negativo) e informações adicionais como por exemplo a identificação do evento que deverá ser acionado no processo lógico receptor. Mesmo depois de executada, uma mensagem continua a fazer parte da lista. Essas mensagens são armazenadas em ordem não decrescente da marca de tempo e são denominadas mensagens positivas;
- Lista de saída de mensagens (*Output Queue*): contém uma cópia de todas as mensagens enviadas pelo processo. Nessa lista todas as mensagens têm marca de tempo menor ou igual a *LVT* e são denominadas mensagens negativas (antimensagens). Essa lista é utilizada em caso de *rollback;*
- Lista de estados (*State Queue*): armazena cópia dos estados recentes do processo lógico. Para viabilizar a execução do *rollback*, o estado do processo é salvo periodicamente.

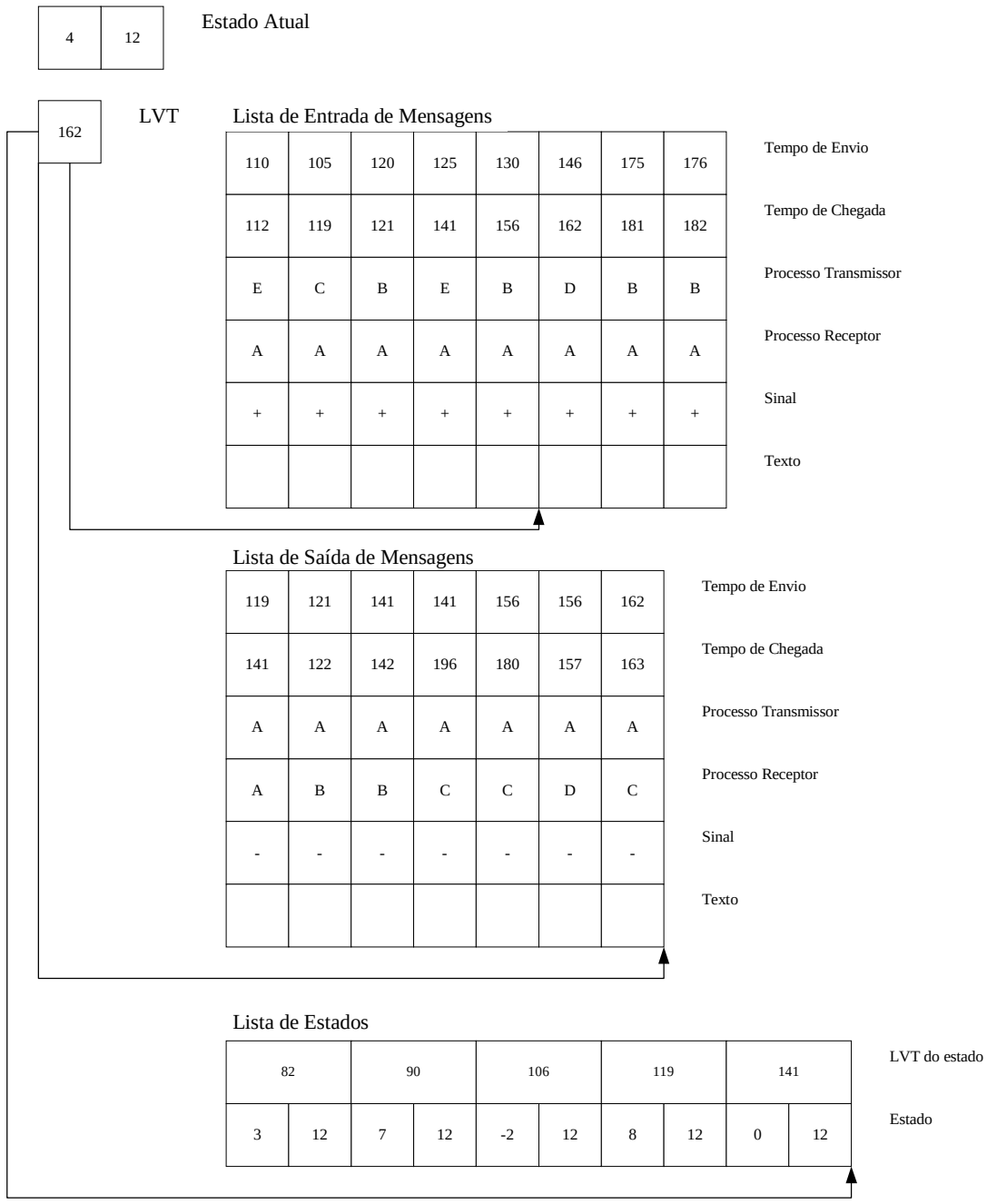


Figura 2.4: Um Processo Lógico *Time Warp*.

2.4.1 Mecanismo de Controle Local

O mecanismo de controle local é responsável pela execução dos eventos da simulação na ordem correta da marca de tempo desses eventos.

Como na simulação seqüencial, um processo lógico repetidamente executa o evento com menor marca de tempo armazenado na *Input Queue*, sem possuir garantias de que a execução desse evento é não irá provocar erros de causa e efeito. O otimismo do protocolo consiste em permitir que os processos lógicos executem os eventos até a ocorrência de algum erro de causa e efeito, verificado quando o processo recebe uma mensagem com marca de tempo menor que o *LVT* do processo. A mensagem causadora do erro de causa e efeito é denominada *straggler*. A figura 2.5 ilustra essa situação, com a chegada da mensagem *straggler* com marca de tempo 18, indicando a ocorrência de um erro de causa e efeito (Fujimoto 2000). No exemplo simplificado, os eventos com marca de tempo 12, 21 e 35 já foram executados, e o *LVT* do processo assume o valor 35. O evento com marca de tempo 41 não foi processado.

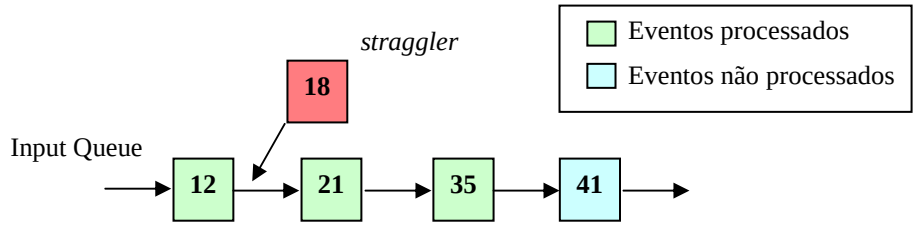


Figura 2.5: Eventos em um Processo Lógico e Chegada de *Straggler*.

A simulação deve então, retornar a um estado consistente. Nesse caso, deve-se desfazer os efeitos de todos os eventos que tenham sido executados prematuramente pelo processo lógico (no exemplo da figura 2.4, os eventos com marcas de tempo 21 e 35 foram processados incorretamente).

Efetuar o *rollback* é a maneira pela qual o *Time Warp* desfaz as computações errôneas que porventura algum processo lógico 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. Isso é efetuado através do envio de antimensagens, que são versões negativas das mensagens (positivas) enviadas por um processo. Se um processo lógico 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 1990a).

A figura 2.6 apresenta o estado do processo lógico antes e após a execução do *rollback* causado pelo evento com marca de tempo 18. Os eventos com marca de tempo 21 e 35 são restaurados utilizando as informações da lista de estados, sendo marcados como não processados. No exemplo, o evento com marca de tempo 35 gerou uma mensagem com marca de tempo 42, então a antimensagem com essa marca de tempo é removida da *Output Queue* e enviada.

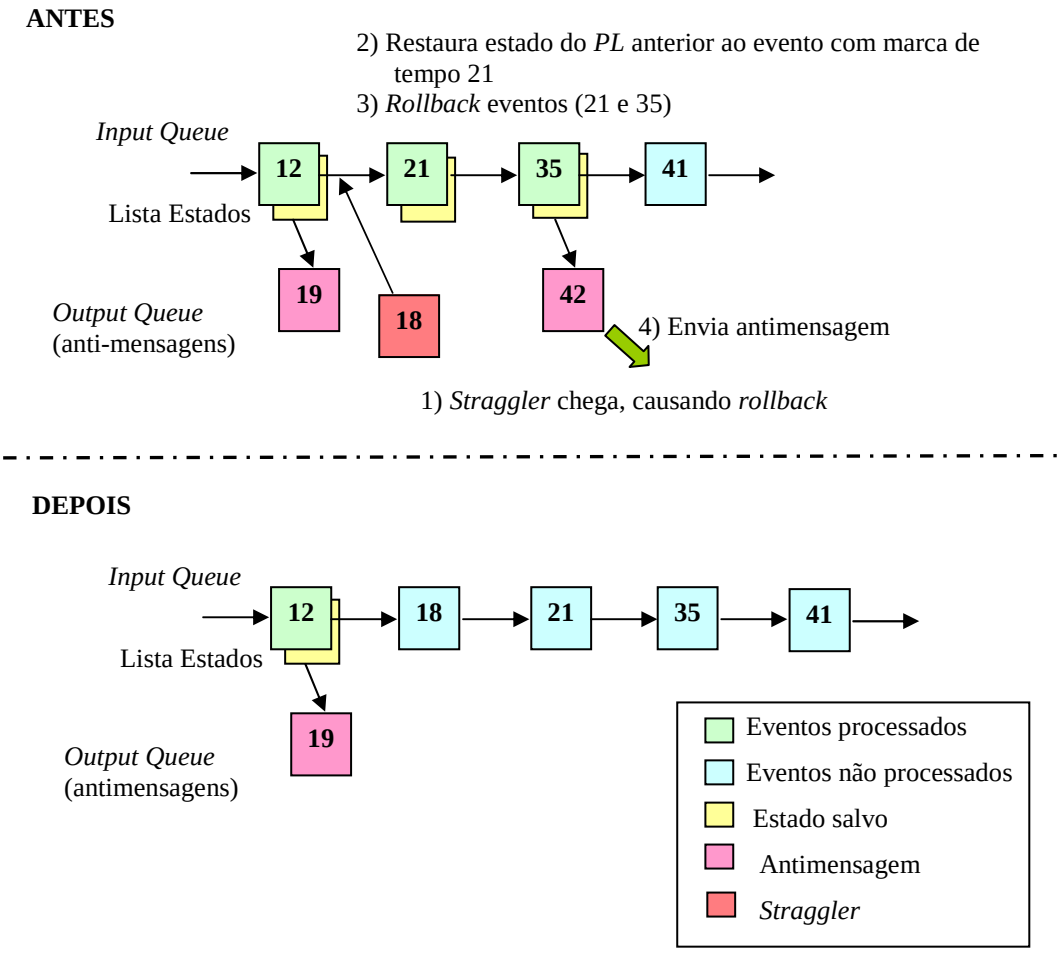


Figura 2.6: *Rollback* Causado por *Straggler*.

Quando um *rollback* ocorre pode existir a necessidade de cancelar algumas mensagens que foram enviadas para outros processos. Para efetuar essa operação podem ser utilizados diferentes tipos de estratégias de cancelamento (Reiher et al. 1990, Kalantery 1997):

- Cancelamento Agressivo
- Cancelamento Preguiçoso;
- Cancelamento Dinâmico;
- Cancelamento Experimental.

O mecanismo *Time Warp* utiliza Cancelamento Agressivo, ou seja, quando um processo lógico 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*. A seção 3.5.1.4 aborda todos os mecanismos de cancelamento de mensagens com maior nível de detalhamento.

Um problema sério relacionado com o *Time Warp* é a necessidade de salvar periodicamente o estado de cada processo lógico, pois o *overhead* de efetuar esta operação pode degradar o desempenho do mecanismo

O protocolo *Time Warp* requer uma quantidade maior de memória para executar uma simulação, em relação à simulação seqüencial correspondente. Na simulação seqüencial é necessário manipular apenas os eventos pendentes (aqueles que estão na lista de eventos futuros). No *Time Warp* o mecanismo de *rollback* exige que os processos salvem seus estados periodicamente, o que introduz sobrecarga no gerenciamento de memória.

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 e exige mecanismos que ajudem a minimizar a utilização do espaço de memória disponível. No capítulo 3 esses algoritmos são apresentados com maiores detalhes.

2.4.2 Mecanismo de Controle Global

O mecanismo de controle global é responsável pelo gerenciamento de memória e detecção do fim da simulação.

Nesse mecanismo, o conceito mais importante é o de *Global Virtual Time* (*GVT*), cujo valor corresponde ao menor valor calculado entre todos os *LVTs* da simulação e todas as marcas de tempo das mensagens enviadas e que estão em trânsito. 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 (Hagenauer 1999). Vários algoritmos têm sido propostos para calcular o *GVT* (Fujimoto 1990b, Fujimoto 1995). Além disso, são necessárias técnicas para cancelar mensagens já enviadas para outros processos. Mais detalhes sobre esse assunto serão apresentados no capítulo 3.

2.5 Redes de Petri

As Redes de Petri consistem em uma técnica de modelagem que permite a representação de sistemas, utilizando como alicerce uma forte base matemática. Essa técnica possui a particularidade de permitir modelar sistemas paralelos, concorrentes, assíncronos e não-determinísticos (Petri 1966, Jensen 1998).

A representação gráfica de uma rede de Petri básica é formada por dois componentes: um ativo chamado de transição (barra) e outro passivo denominado lugar (círculo). Os lugares eqüivalem às variáveis de estado (possíveis situações do sistema) e as transições correspondem às ações realizadas pelo sistema. Esses dois componentes são ligados entre si através de arcos dirigidos. Os arcos podem ser únicos ou múltiplos. A figura 2.7 mostra os elementos básicos de um grafo associado às redes de Petri.

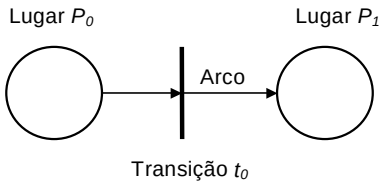


Figura 2.7: Grafo e seus Elementos Básicos.

2.5.1 Definições

As redes de Petri podem ser enfocadas através de três fundamentações diferentes. A primeira utiliza a teoria *bag* como suporte. A segunda usa os conceitos da álgebra matricial. A terceira se fundamenta na estrutura definida por relações. A seguir são apresentadas as definições formais de cada uma dessas fundamentações. Uma abordagem mais detalhada do assunto pode ser encontrada em (Francês et al. 2000).

- Definição 1: uma rede de Petri **R** é uma ênupla ordenada R = (P, T, I, O, K), onde P = $\{p_1, p_2,...,p_n\}$ é um conjunto finito não-vazio de lugares, T = $\{t_1, t_2,..., t_m\}$ é um conjunto finito não-vazio de transições. I : T $\rightarrow$ P é um conjunto de *bags* [1] que representa o mapeamento de transições para lugares de entrada. O : T $\rightarrow$ P é um conjunto de *bags* que representa o mapeamento de transições para lugares de saída. K : P $\rightarrow$ N é o conjunto da capacidades associadas a cada lugar, podendo assumir um valor infinito (Peterson 1981);
- Definição 2**:** a estrutura de uma rede de Petri, segundo o ponto de vista matricial, é uma quíntupla R = (P, T, I, O, K), onde P é um conjunto finito de lugares. T é um conjunto finito de transições, I : P x T $\rightarrow$ N é a matriz de pré-condições. O : P x T $\rightarrow$ N é a matriz de pós-condições. K é o vetor das capacidades associados aos lugares (K : P $\rightarrow$ N) (Peterson 1981);
- Definição 3: a estrutura de redes de Petri, usando-se relações, é formada por uma quíntupla R = (P, T, A, V, K), onde P é o conjunto de lugares, T o de transições, A o conjunto dos arcos e V corresponde ao conjunto de valorações desses arcos. Os elementos de A são arcos que conectam transições a lugares ou lugares a transições (A $\subseteq$ (P x T) $\cup$ (T x P)). Assim, os elementos de A podem ser agrupados em dois subconjuntos - o conjunto das entradas às transições e o de saída às transições, I = $\{(p_i, t_j)\}$ e O = $\{(t_j, p_i)\}$, respectivamente (Murata 1989).

[1] *Bag* é uma generalização do conceito de conjunto que admite a repetição de elementos. Na notação de *bags*, utiliza-se [ ], enquanto que para denotar conjuntos, utiliza-se { } (Maciel et al., 1996).

2.5.2 Redes de Petri Marcadas

Marcas (*tokens*) são informações atribuídas aos lugares, para representar a situação (estado) da rede em um determinado momento. Define-se uma rede de Petri marcada pela dupla RM = (R, M_o), onde R é a estrutura da rede e M_o a marcação inicial (Maciel et al. 1996). Assim, para simular o comportamento dinâmico dos sistemas, a marcação da rede de Petri é modificada a cada ação realizada (transição disparada). A figura 2.8 ilustra uma rede marcada.

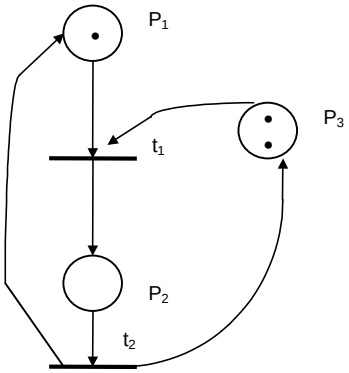


Figura 2.8: Rede Marcada.

2.5.3 Notações Particulares

Em alguns casos, deseja-se representar a diferença entre transições, visando melhorar a clareza do modelo. Além disso, em muitas situações, pretende-se representar a execução de uma condição externa ao sistema modelado. Para representar rótulos de transições, utiliza-se um alfabeto qualquer associado à rede (por exemplo, o alfabeto a,b,c,....,z). Para representar as condições externas, usa-se o mesmo esquema utilizado para rotular transições, entretanto, os símbolos vêm entre parênteses (conforme ilustra a figura 2.9).

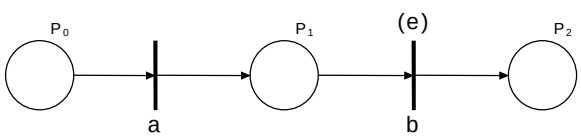


Figura 2.9: Rótulos e Condições Externas às Transições.

2.5.4 Classes de Redes de Petri

Podem-se agrupar as redes de Petri em duas grandes classes: Ordinárias e Não-Ordinárias (de Alto nível) (Maciel et al. 1996). As redes ordinárias utilizam apenas um tipo básico de marcas, o tipo inteiro não-negativo. Fazendo-se uma analogia às linguagens de programação, as redes de alto nível podem possuir marcas mais sofisticadas, como tipos de dados definidos pelo usuário ou ainda tipos compostos que são formados por vários tipos mais elementares (por exemplo, na linguagem C, *Structs*). As redes ordinárias se subdividem em:

- Rede Binária: é a rede mais elementar dentre todas. Essa rede permite, no máximo, um *token* em cada lugar, e todos os arcos possuem valor unitário;
- Rede *Place-Transition*: é o tipo de rede que permite o acúmulo de marcas no mesmo lugar, assim como valores não unitários para os arcos.

As redes de alto nível se diferenciam das ordinárias basicamente por individualizarem os *tokens*. Assim, por exemplo, em um mesmo lugar "fruteira" pode haver os *tokens* maçã, banana e laranja (cada um de um tipo diferente). Esse tipo de rede permite a individualização de uma marca (pertencente a um grupo) em um mesmo lugar. Essa individualização pode ser realizada através de vários artifícios, como por exemplo, cor da marca ou objetos representando os *tokens*. Redes não-ordinárias permitem uma maior clareza e um maior nível de abstração ao modelo.

2.5.5 Redes Elementares

Nesta seção, são apresentadas algumas redes, a partir das quais se derivam outras redes mais complexas. São discutidas as redes representativas de seqüenciamento, distribuição, junção e escolha não-determinística.

- Seqüenciamento: é a rede que representa a execução de uma ação, desde que uma determinada condição seja satisfeita. Após a execução dessa ação, pode-se ter outra ação, desde que seja satisfeita uma determinada condição (figura 2.10).

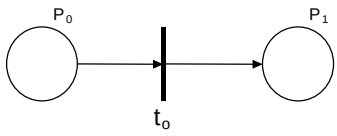


Figura 2.10: Seqüenciamento.

- Distribuição: é a rede elementar utilizada na criação de processos paralelos a partir de um processo pai. Os processos filhos são criados através da distribuição dos *tokens* encontrados no processo (lugar) pai. A distribuição é mostrada na figura 2.11. É importante observar que se houvesse um *token* em P_1, ele seria propagado (obrigatoriamente) tanto para P_2 quanto para P_3.

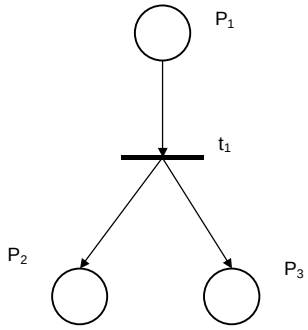


Figura 2.11: Distribuição.

- Junção: é a rede usada para sincronizar atividades concorrentes. No exemplo da figura 2.12, a transição t_1 só dispara quando existirem marcas tanto em P_1 quanto em P_2, estabelecendo, assim, o sincronismo.

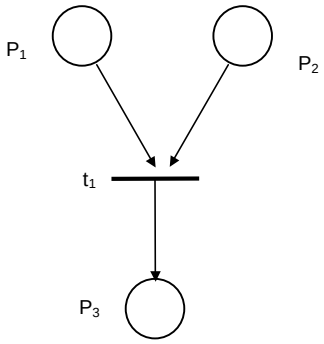


Figura 2.12: Junção.

- Escolha Não-Determinística: nessa rede, o disparo de uma transição inabilita a outra (figura 2.13). O fator não-determinístico dessa rede gera uma situação chamada de conflito. O conflito pode ser classificado como estrutural ou efetivo. Ambos os conflitos estão associados ao fato de duas transições possuírem o mesmo lugar como entrada. Porém, se a rede não possuir *tokens*, o conflito é dito estrutural. Contudo, se há uma única marca no lugar comum às transições, diz-se que o conflito é efetivo. A figura 2.14 (Maciel et al. 1996) ilustra os dois tipos de conflito.

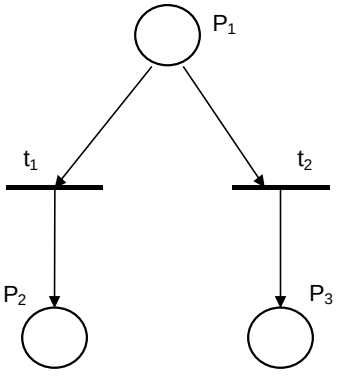


Figura 2.13: Escolha Não-Determinística.

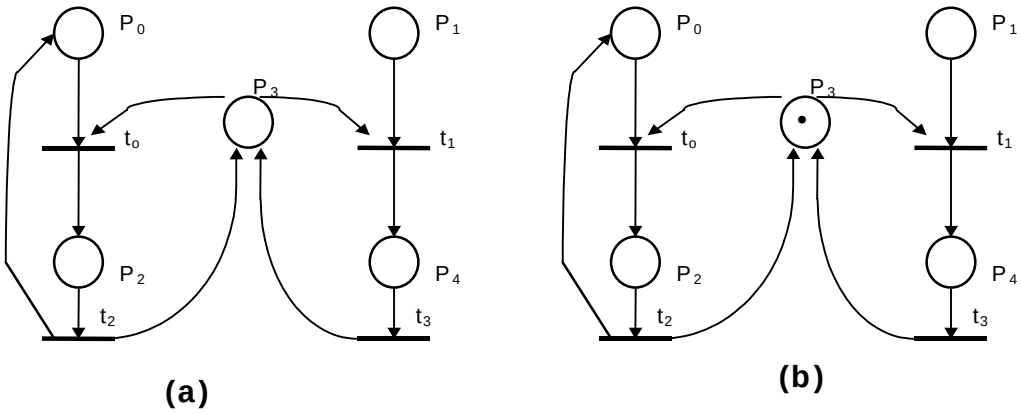


Figura 2.14. (a) Conflito Estrutural e (b) Conflito Efetivo.

A escolha determinística da transição a ser disparada não é um recurso abordado nas redes elementares. Porém, essa deficiência das redes de Petri originais é resolvida em algumas extensões propostas, abordadas em seções posteriores.

2.5.6 Extensões às Redes de Petri

Nesta seção são vistas algumas extensões propostas com a finalidade de aumentar a aplicação das redes de Petri. São discutidas as redes de Petri com arco inibidor, coloridas, hierárquica e temporizadas determinísticas.

2.5.6.1 Redes de Petri com Arco Inibidor

Observando-se a figura 2.15, pode-se considerar que se deseja estabelecer uma ordem de disparo, isto é, T1 precede T2. Para essas situações pode-se utilizar o recurso do arco inibidor (arco com um pequeno círculo na extremidade), conforme mostra a figura 2.16.

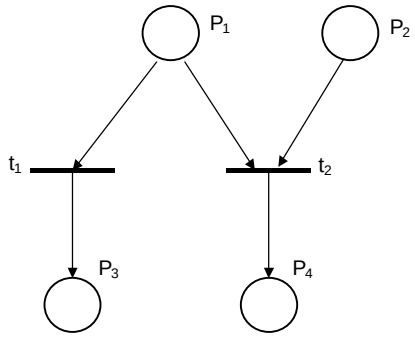


Figura 2.15: Não-determinismo no Disparo de T1 e T2.

A nova regra de disparo é que as transições que têm como entrada arcos inibidores só serão habilitadas quando, em seus lugares de entrada, não houver mais *tokens*. Essa condição é chamada de teste de zero marcas. Assim, retomando-se o exemplo da figura 2.15, a transição T2 só será habilitada quando, no lugar P1, não houver mais nenhuma marca.

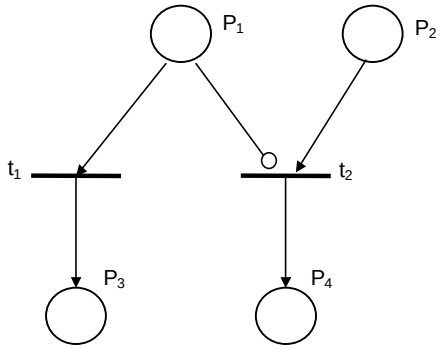


Figura 2.16: Rede com Arco Inibidor.

2.5.6.2 Redes de Petri Coloridas

As redes de Petri coloridas têm por objetivo reduzir o tamanho do modelo, permitindo que os *tokens* sejam individualizados, através de cores atribuídas a cada um deles. Assim, diferentes clientes, processos ou recursos podem ser representados em uma mesma rede, o que é bastante conveniente em modelos que representam eficientemente sistemas reais. As cores não significam apenas cores ou padrões. Elas podem representar tipos de dados complexos, usando a nomenclatura de colorida apenas para referenciar a possibilidade de distinção entre os *tokens* (Jensen et al.

1990). A figura 2.17 apresenta uma rede colorida onde os arcos são rotulados com cores (DNS, Telnet, FTP, HTTP) (Francês et al. 2000).

Nesse exemplo, há um cliente TCP/IP que pode solicitar quatro tipos de serviços a quatro servidores diferentes (o servidor DNS, o servidor Telnet, o servidor FTP e o servidor HTTP). Então, as solicitações do cliente devem ser diferenciadas, de acordo com o servidor que vai atendê-las. A associação de cores e marcas individualizadas para cada requisição, e as restrições colocadas nos arcos determinam quais marcas podem atravessar esses arcos. Na figura 2.17, utiliza-se o modo mais elementar de redes coloridas, no qual se associa ao arco uma determinada cor. Assim, o *token* se destinará ao arco cuja cor for idêntica a da marca. Desse modo, pode-se perceber que os *tokens* de "Cliente" não habilitarão as transições t_1 e t_2, pois os arcos que ligam Cliente às transições t_1 e t_2 só aceitam cores do tipo Telnet e FTP, e o lugar Cliente só possui marcas do tipo DNS e HTTP. O que significa dizer que o cliente só pode solicitar serviços de DNS e HTTP.

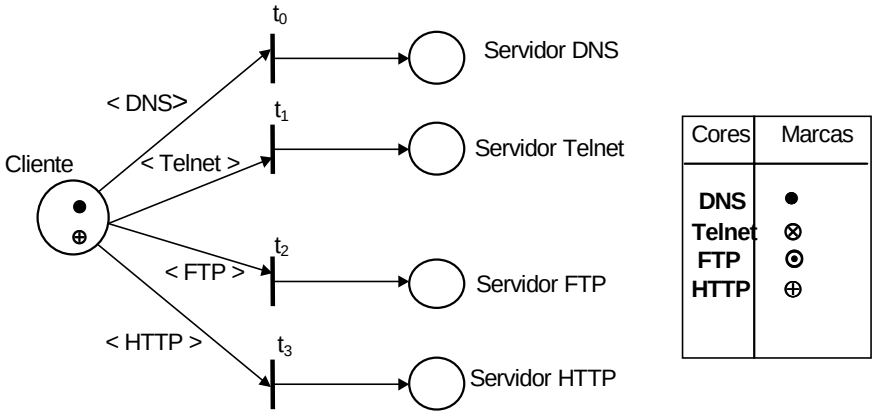


Figura 2.17: Rede de Petri Colorida.

2.5.6.3 Redes Hierárquicas

Um dos problemas apresentados pelasas redes de Petri originais é o fato de que, conforme o tamanho do sistema cresce, vai se tornando cada vez mais difícil manter a clareza do modelo. Esse fato se deve, em grande parte, à falta de hierarquização dos modelos originais. Por exemplo, a figura 2.18 apresenta um protocolo simples, com um transmissor e dois receptores. As representações de redes

ordinárias não permitem a criação de um modelo com maior abstração, apesar de receptores e transmissor poderem ser agrupados em elementos mais gerais.

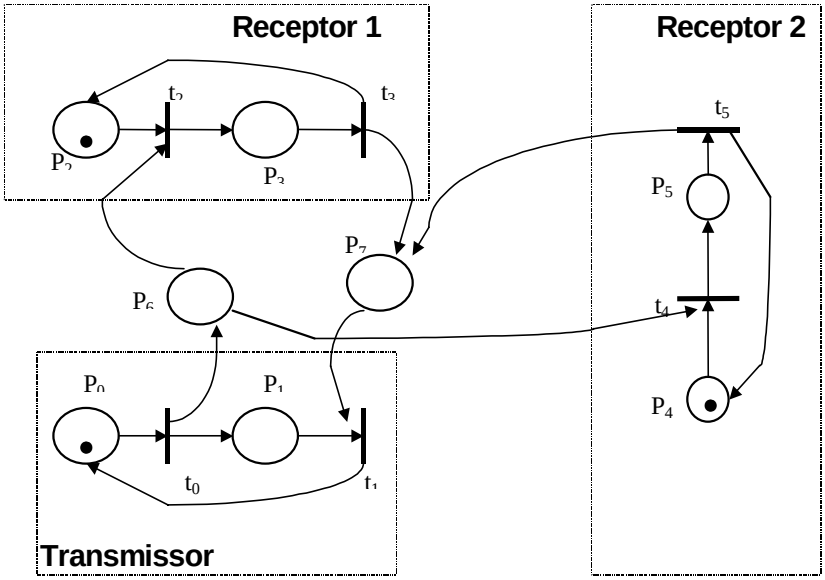


Figura 2.18: Protocolo Simples em Rede de Petri Ordinária.

Para amenizar essa limitação, foram criados mecanismos que possibilitam o agrupamento ou o refinamento de partes do modelo. Um dos problemas dessa abordagem de mais alto nível é manter a consistência com os elementos vizinhos àqueles que sofrem um agrupamento. Na abordagem hierárquica, mostrada nesta seção, lugares e transições podem ser apresentados sob uma ótica mais abstrata.

Na representação hierárquica, dois componentes são fundamentais para viabilizar uma representação em mais alto nível: a superpágina e a subpágina (Dittrich 1995). A primeira representa um agrupamento de componentes (transições, lugares e arcos), visando à geração de um modelo mais compacto e inteligível, como se fosse uma “caixa preta”. Já as subpáginas são o refinamento de uma superpágina, de forma a esclarecer alguns detalhes omitidos na representação em alto nível. A figura 2.19 apresenta uma representação em alto nível para o exemplo do protocolo de comunicação visto na figura 2.18. O exemplo mostra a representação do transmissor e dos receptores 1 e 2 em forma de superpáginas.

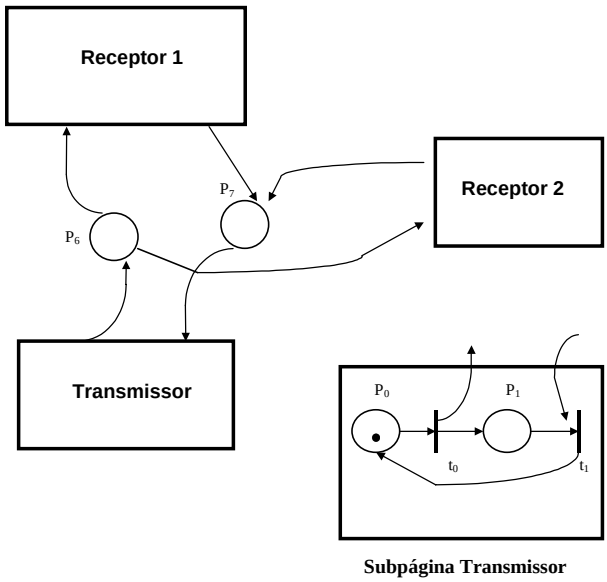


Figura 2.19: Hierarquia Utilizando Superpáginas.

2.5.6.4 Rede de Petri Temporizada

As redes tradicionais não incluem qualquer conceito de tempo e permitem descrever apenas a estrutura lógica de um sistema, não sua evolução no tempo. A introdução do tempo deu origem às Redes Temporizadas, que possibilitam a representação do comportamento dinâmico de sistemas que possuam atividades concorrentes e assíncronas.

A associação do tempo a componentes da rede pode se realizar de várias maneiras. As principais são (Maciel et al. 1996):

- O tempo associado aos lugares: os *tokens* (após o disparo de uma transição) só estarão disponíveis para disparar uma nova transição após um determinado tempo que está associado ao lugar;
- O tempo associado às marcas: nesse caso o tempo indica quando a marca estará disponível para disparar uma transição;
- O tempo associado às transições: o *token* só estará disponível no lugar de saída da transição após o tempo associado.

Um exemplo de rede de Petri temporizada é apresentado na figura 2.20. Nela, as transições t_1 e t_2 possuem tempos associados diferentes, o que significa que uma

transição será disparada antes da outra. Supondo-se que $d_1 < d_2$, então o *token* chegará primeiro ao lugar P_1. Assim, de maneira determinística, pode-se estabelecer a ordem em que os eventos devem ocorrer.

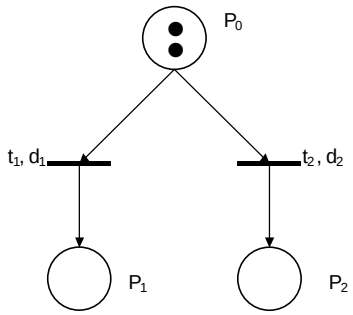


Figura 2.20: Rede de Petri Temporizada.

2.5.6.5 Redes de Petri para Avaliação de Desempenho

As Redes de Petri tradicionais e extensões apresentadas permitem efetuar apenas o estudo comportamental de um sistema, verificando a sintaxe e a semântica do modelo. Para que fosse possível efetuar o estudo de desempenho de um sistema, tornou-se necessário introduzir a noção de processos estocásticos nas Redes de Petri tradicionais (Marsan et al. 1986). As seções a seguir apresentam as características das Redes de Petri Estocásticas e as Redes de Petri Estocásticas Generalizadas.

2.5.6.5.1 Redes de Petri Estocásticas

Modelos probabilísticos de desempenho tentam representar o comportamento de sistemas determinísticos complexos através de processos estocásticos. Dessa forma, para que as Redes de Petri pudessem ser utilizadas para avaliação de desempenho, foram introduzidas mudanças nas Redes de Petri Temporizadas.

A união das características de Redes de Petri (que são voltadas para descrever situações de concorrência e sincronismo) com um modelo estocástico permite realizar o estudo de desempenho de sistemas computacionais complexos, através das Redes de Petri Estocásticas (*Stochastic Petri Nets – SPN*) (Marsan et al. 1998).

Nessa variação das Redes de Petri, associa-se a cada transição uma variável aleatória exponencialmente distribuída, que expressa a quantidade de tempo decorrido até que a transição seja habilitada.

As *SPN* podem ser formalmente definidas como sendo (Marsan 1989):

SPN = (P, T, A, M_0, L) onde:

P = $\{p_1, p_2, ..., p_n\}$ é o conjunto de lugares;

T = $\{t_1, t_2, ..., t_m\}$ é o conjunto de transições;

A = $(A_e \cup A_s)$ é o conjunto de arcos;

$A_e \subset (P \times T)$ é o conjunto de arcos de entrada;

$A_s \subset (T \times P)$ é o conjunto de arcos de saída;

$M_0 = \{m_{01}, m_{02}, ..., m_{0n}\}$ onde m_{0i} representa a número de *tokens* no lugar p_i, na marcação inicial M_0;

L = $\{l_1, l_2, ..., l_m\}$ é o conjunto de taxas de disparo associadas às transições.

As Redes de Petri Estocásticas permitem a obtenção de estatísticas como por exemplo o tempo médio que um *token* gasta para percorrer a rede, o número médio de vezes que uma determinada transição dispara e o número médio de *tokens* em um lugar.

A figura 2.21 apresenta um exemplo de uma Rede de Petri Estocástica que descreve um sistema simples onde clientes chegam segundo um determinado tempo entre-chegadas (transição *Gera_Clientes*), são atendidos pelo único servidor disponível por um determinado tempo (transição *Servidor*) e deixam o sistema (lugar *Serviço_Completo*). Se o servidor está ocupado, os clientes aguardam no lugar *Fila*. A capacidade do centro de serviços é determinada pela presença de *tokens* no lugar *Capacidade_do_Servidor*. Como exemplo de estatística que pode ser obtida, pode-se pensar no comprimento médio da fila, que é igual ao número médio de *tokens* que permanecem no lugar *Fila*.

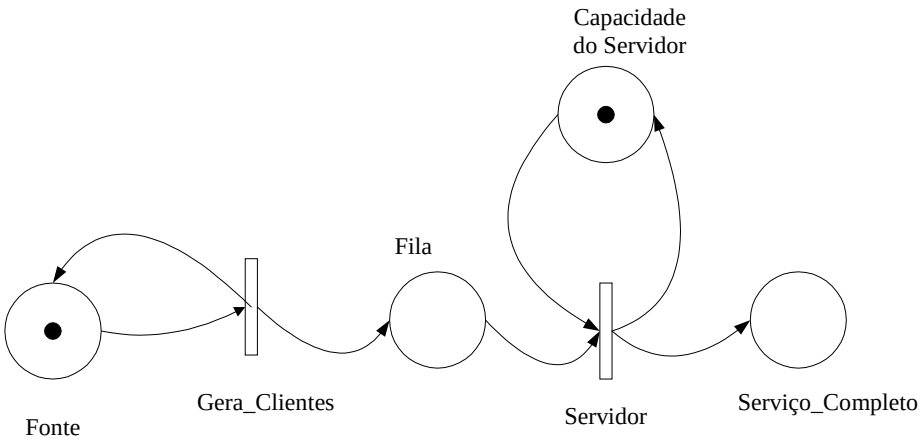


Figura 2.21: Um Exemplo de Rede de Petri Estocástica.

2.5.6.5.2 Redes de Petri Estocásticas Generalizadas

A principal motivação para o desenvolvimento das Redes de Petri Estocásticas Generalizadas (*Generalized Stochastic Petri Nets - GSPN*) consiste no fato de que em determinadas situações não é desejável associar um tempo aleatório a todas as transições, mas sim apenas àquelas que estão relacionadas com eventos que exerçam maior impacto no desempenho do sistema em estudo (Marsan et al. 1998).

Dessa forma, as *GSPN* possuem transições imediatas, que disparam em tempo zero quando estão habilitadas, e transições temporizadas, que disparam após um tempo aleatório exponencialmente distribuído quando estão habilitadas. As transições imediatas têm prioridade sobre as transições temporizadas. Para facilitar a distinção, convenciona-se utilizar um traço para representar graficamente uma transição imediata e um retângulo para representar uma transição temporizada (figura 2.22).

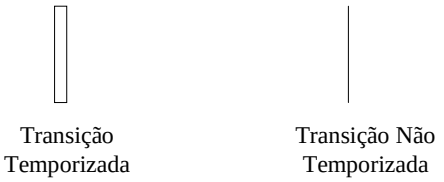


Figura 2.22: Representação Gráfica de Transições.

Formalmente, uma Rede de Petri Estocástica Generalizada pode ser representada por:

GSPN = (P, T, A, M_0, L) onde:

P = {p_1, p_2, ..., p_n} é o conjunto de lugares;

T = {t_1, t_2, ..., t_m} é o conjunto de transições;

A = ($A_e \cup A_s$) é o conjunto de arcos;

$A_e \subset$ (P x T) é o conjunto de arcos de entrada;

$A_s \subset$ (T x P) é o conjunto de arcos de saída;

M_0 = {m_{01}, m_{02}, ..., m_{0n}} onde m_{0i} representa a número de *tokens* no lugar p_i, na marcação inicial M_0;

L = {l_1, l_2, ..., $l_{m'}$} é o conjunto de taxas de disparo associadas as transições e m' é o número de transições temporizadas.

A figura 2.23 apresenta a estrutura da arquitetura de um sistema multiprocessador (Marsan et al. 1998). Os elementos de processamento, que contêm memória local, são conectados por um barramento, o qual permite acesso à memória compartilhada. Após a execução de uma tarefa (tempo aleatório), os elementos de processamento requisitam acesso à memória compartilhada. Se o barramento não está disponível os elementos de processamento aguardam. A memória e o barramento são liberados após a utilização da memória (tempo aleatório) e o elemento de processamento retorna para suas atividades. No desenvolvimento do modelo em Redes de Petri Estocástica Generalizada da figura 2.24, o autor desconsiderou os tempos associados a arbitragem do barramento (transições temporizadas após os lugares *Espera pelo barramento*).

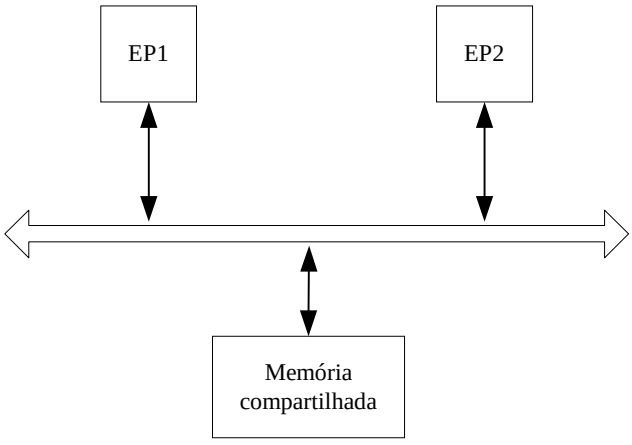


Figura 2.23: Arquitetura de um Multiprocessador.

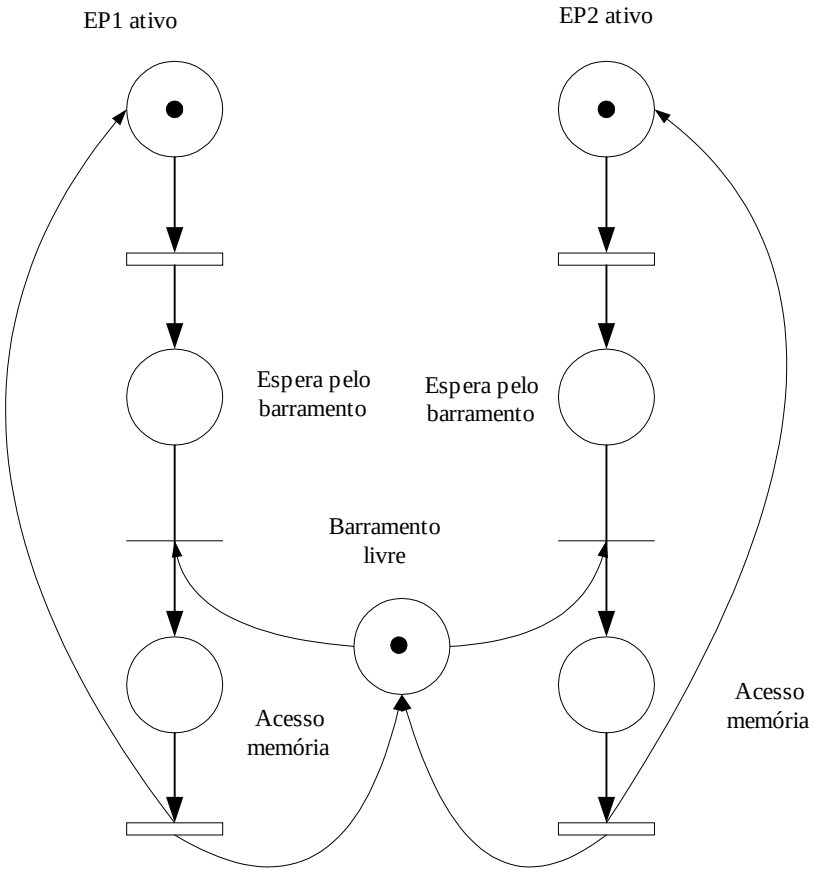


Figura 2.24: Modelo em Rede de Petri Estocástica Generalizada da Figura 2.23.

2.6 Considerações Finais

A abordagem para simulação seqüencial utiliza uma estrutura de dados denominada lista de eventos futuros e uma variável relógio que marca a passagem do tempo na simulação. A natureza inerentemente seqüencial da lista de acontecimentos futuros constitui um obstáculo a ser superado quando se deseja particionar a simulação e executá-la em mais de um elemento de processamento. O principal problema relacionado com a simulação distribuída consiste em lidar com a ocorrência de erros de causa e efeito ou seja, impedir que o futuro possa afetar o passado. Do ponto de vista do sistema físico, a causa sempre deve preceder o efeito, e é responsabilidade do protocolo de sincronização da simulação garantir esse sequenciamento quando o programa de simulação é executado de forma distribuída.

Os protocolos conservativos e otimistas solucionam esse problema a partir de diferentes abordagens. As abordagens conservativas evitam a possibilidade de ocorrência de erros de causa e efeito, enquanto nos protocolos otimistas a simulação progride até encontrar um erro de causa e efeito, e um mecanismo de *rollback* é utilizado para restaurar a simulação para um estado consistente.

Este capítulo apresentou uma descrição do problema de sincronização em simulação distribuída, enfatizando a solução através dos protocolos otimistas que têm como principal representante o *Time Warp*, o qual é objeto de estudo desta tese. Foi apresentada também uma síntese sobre os principais conceitos relacionados a utilização das Redes de Petri e sua aplicação na avaliação de desempenho. A bibliografia sobre os protocolos de sincronização para simulação distribuída e sobre Redes de Petri é vasta e uma revisão mais completa sobre o assunto é apresentada em (Spolon et al. 2000b).

A contribuição deste capítulo para esta tese está no fornecimento de subsídios básicos para o estudo das características importantes de um processo lógico *Time Warp* e sua representação através das Redes de Petri (que será detalhado no capítulo 4). Existem outras técnicas mas optou-se pelas Redes de Petri devido às suas características (esse tópico é abordado no capítulo 4).

No capítulo 3 é apresentada uma discussão sobre os protocolos de sincronização otimistas que têm como padrão o uso de antimensagens e *rollbacks*, como no *Time Warp*. A partir desse estudo define-se a proposta de uma nova taxonomia que permite organizar e visualizar esses protocolos de forma hierárquica.

Capítulo 3

Proposta de Taxonomia Hierárquica para Protocolos Otimistas

Este capítulo discute os principais protocolos variantes do *Time Warp* e as classificações existentes. Como contribuição apresenta uma nova proposta de taxonomia concisa e hierárquica que facilita o trabalho de avaliação e comparação dos protocolos que limitam o otimismo no *Time Warp*.

3.1 Considerações Iniciais

O protocolo conservativo *CMB* e o protocolo otimista *Time Warp* são dois protocolos de sincronização em simulação distribuída discutidos amplamente na literatura nos últimos 20 anos.

Não existe um consenso sobre qual dos dois apresenta melhor desempenho e por esse motivo tem-se optado por desenvolver protocolos que buscam explorar as vantagens que os protocolos *CMB* e *Time Warp* apresentam. Assim, podem-se inserir características conservativas em um protocolo puramente otimista, ou então tornar mais otimista um protocolo conservativo. A primeira opção, por exemplo, deu origem a muitas variantes do protocolo otimista *Time Warp*.

Um novo problema surgiu para aqueles que procuravam uma alternativa entre *CMB* e *Time Warp*, envolvendo a tomada de decisão sobre qual dos protocolos otimistas utilizar para a resolução de problemas. Uma avaliação entre esses protocolos fica facilitada quando se consegue organizá-los, agrupando os protocolos

que possuem características semelhantes. Com uma classificação desse tipo, pode-se então avaliar um grupo ao invés de um protocolo específico.

As classificações apresentadas na literatura são redundantes e confusas, o que levou à proposta de uma nova taxonomia para facilitar o estudo dos protocolos otimistas que inserem noções de conservadorismo no *Time Warp*. A proposta inicial desta nova taxonomia foi publicada nos anais da *13th European Simulation Multiconferece* (ESM'2000) (Spolon et al. 1999) e é revisada neste capítulo. Através dessa taxonomia busca-se preencher a lacuna existente entre os protocolos e a avaliação de desempenho que será discutida no capítulo 4.

A seção 3.2 reúne os principais protocolos que modificam o *Time Warp* destacando suas características. A seção 3.3 discute a necessidade de se modificar o *Time Warp* no sentido de reduzir o número de computações executadas de forma errônea, apresentando também as classificações de protocolos que efetuam essa tarefa. A seção 3.4 resume os principais protocolos que limitam otimismo no *Time Warp* e também aqueles referenciados nas classificações da seção 3.3, fornecendo tabelas comparativas, onde cada protocolo é inserido nas classificações existentes. A seção 3.5 propõe uma nova taxonomia para os protocolos que limitam o otimismo no *Time Warp*, apresentando as vantagens em relação às classificações existentes. A seção 3.6 apresenta as considerações finais.

3.2 Limitação do Otimismo no *Time Warp*

Esta seção descreve os principais protocolos que alteram o *Time Warp* de forma a controlar seu excesso de otimismo (uma revisão completa e detalhada pode ser encontrada em (Spolon et al. 2000b)). Verificar as principais características de cada protocolo é o primeiro passo para a proposta da nova taxonomia, como será apresentado na seção 3.5.

3.2.1 *Breathing Time Buckets* (*BTB*)

Esse protocolo é utilizado no ambiente *SPEEDES (Synchronous Parallel Environment for Emulation and Discrete Event Simulation*). *BTB* utiliza processamento otimista com *rollback* local. Os eventos são processados de forma

otimista, mas as mensagens geradas por eventos são liberadas somente quando se tem certeza de que são válidas (Steinman 1991, Steinman 1992, Steinman 1993, Steinman 1995).

BTB processa os eventos em ciclos, 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 a tarefa de *rollback* consiste simplesmente em restaurar o estado do processo lógico e descartar as mensagens que foram geradas de forma incorreta (Steinman 1992).

3.2.2 *Breathing Time Warp* (*BTW*)

A motivação para o desenvolvimento desse protocolo é o fato de que o *Time Warp* é extremamente otimista, como por exemplo em simulações onde a granulosidade dos eventos é fina (a sobrecarga de comunicação é significativa, portanto 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 (Steinman 1992, Steinman 1995).

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 protocolo consiste em uma combinação de *Time Warp* e *BTB.*

3.2.3 *Local Time Warp* (*LTW*)

Combina de forma hierárquica os protocolos *Conservative Time Windows* (Ayani; Rajaei 1992, Ayani; Rajaei 1994) e *Time Warp*. O sistema consiste em um número de subsistemas (grupos), compostos por vários processos lógicos (Rajaei et al. 1993). Dentro dos grupos, utiliza *Time Warp* para explorar melhor o paralelismo da aplicação. Para impedir que *rollbacks* de um grupo afetem outro grupo, utiliza *Conservative Time Windows* para a sincronização entre os mesmos.

3.2.4 *Bounded Time Warp*

O protocolo *Bounded Time Warp* divide o tempo de simulação em intervalos de tamanhos iguais, sendo similar ao *Moving Time Windows* (seção 3.2.7) (Das 1996, Rajaei et al. 1993).

O ponto central desse protocolo consiste em um algoritmo que pode detectar quando todos os processos lógicos 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 processo lógico (o mestre, inicialmente) coloca o *token* no final de sua fila de eventos. 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 processo lógico está inativo. O processo lógico corrente passa o *token* para o próximo e assim sucessivamente, formando um anel lógico (Turner; Xu 1992a, Turner; Xu 1992b).

3.2.5 *Filter*

Filter é um protocolo otimista onde 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 processo lógico divulga, através de difusão, informações 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).

3.2.6 *SFilter* (*Single-hop Filter*)

Esse protocolo consiste em uma variação do protocolo *Filter*. Cada processo lógico 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 difusão global, mas local. Essa limitação da propagação da informação pode reduzir a sobrecarga por mensagem (Prakash; Subramanian 1992).

3.2.7 *Moving Time Windows* (*MTW*)

Nesse protocolo, uma janela de tempo global, dinamicamente ajustada, é atribuída aos processos lógicos (Ayani; Rajaei 1994). Dois eventos *e* e *e'* com tempos de ocorrência *ot(e)* e *ot(e')* podem ser simulados em paralelo somente se $|ot(e) - ot(e')| < \Delta$ onde Δ é o tamanho pré-definido da janela (Ferscha 1995a).

Esse protocolo deu origem aos trabalhos que procuram limitar a ocorrência de *rollbacks* no *Time Warp*, limitando assim seu otimismo (Das 1996).

3.2.8 *Adaptive Time Warp Concurrency Control Algorithm* (*ATW*)

Esse protocolo 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 uma janela de tempo denominada *Blocking Window* (*BW*) e somente os eventos com marcas de tempo dentro da janela podem ser executados. O tamanho da *BW* adaptativa é determinado com base 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, Ferscha 1995b).

3.2.9 *Probabilistic Distributed discrete event Simulation Protocol (PDSP)*

O objetivo 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 protocolo adapta probabilisticamente o tamanho da *LVTW*, de acordo com a variação das marcas de tempo dos eventos que chegam (Ferscha; Chiola 1994).

3.2.10 *Probabilistic Direct Optimism Control* (*PADOC*)

Nesse protocolo cada processo lógico monitora o progresso do *LVT*, registrando as marcas de tempo das mensagens que chegam. Com base no padrão de chegada observado, cada processo lógico formula uma hipótese sobre a marca de tempo da próxima mensagem esperada e, dependendo do valor previsto, adapta-se buscando um comportamento de sincronização que busca ser o melhor entre *CMB* e *Time Warp*. O protocolo combina características do *Probabilistic Distributed Discrete Event Simulation Protocol* e do *Adaptive Time Warp* (Ferscha 1995d).

Esse protocolo bloqueia temporariamente o processamento dos eventos, evitando a geração e envio de mensagens em estados para os quais é provável que tenham que ser desfeitos. A análise estatística do histórico das chegadas de mensagens é utilizada para fazer previsões para as marcas de tempo das mensagens futuras, permitindo a cada processo lógico adaptar localmente seu comportamento em relação à estratégia de sincronização mais eficiente (Ferscha 1995a).

3.2.11 *MIMDIX*

Nesse protocolo, em pontos discretos do tempo, cada processo lógico envia, com certa probabilidade, uma mensagem de re-sincronização aos demais processos lógicos. 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 ordem cronológica da simulação (Madisseti et al. 1992).

Em intervalos de tempo δ determinados probabilisticamente, os processos lógicos 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, uma mensagem de difusão *SYNC* (com marca de tempo um pouco maior do que *GVT*) para o sistema, impedindo que os processos lógicos avancem muito no tempo simulado (Ferscha 1995a, Rajaei et al. 1993).

A re-sincronização é executada em cada processo lógico 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.

3.2.12 Algoritmos Adaptativos *NPSI*

Os protocolos adaptativos *Near Perfect State Information* (*NPSI*) são protocolos nos quais o otimismo é controlado dinamicamente através de informações obtidas a partir do estado da simulação distribuída (Srinivasan; Reynolds 1995a, Srinivasan; Reynolds 1995b).

3.2.13 *Unified Distributed Simulation* (*UDS*)

Esse protocolo utiliza um mecanismo de janelas para envio e recebimento de mensagens. As mensagens com marcas de tempo fora dos limites das janelas não são enviadas ou recebidas, mas os processos lógicos são bloqueados, o que pode levar a um *deadlock* (Rajaei et al. 1993).

3.2.14 *WOLF*

Nesse protocolo a ocorrência de *rollback* implica no envio de mensagens especiais em forma de difusão, 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).

3.2.15 *Window-based Throttling*

Nesse protocolo um processo lógico somente pode executar as mensagens de eventos cujas marcas de tempo estão dentro de uma janela de tempo (Reynolds 1989).

3.2.16 *Penalty-based Throttling*

Outro método de limitar o otimismo no *Time Warp* é manter controle sobre quais processos lógicos estão fazendo trabalho correto e quais estão fazendo trabalho que deverá ser descartado. Os primeiros são beneficiados pelo algoritmo de escalonamento de processos do elemento de processamento e têm mais ciclos de UCP 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).

3.2.17 *Adaptive Bounded Time Window*

Esse protocolo é similar ao *Moving Time Window* (seção 3.2.7). A largura da janela é controlada dinamicamente, com base no conceito de trabalho útil executado por um processo lógico. O trabalho útil corresponde a uma medida do trabalho produtivo executado por um processo lógico, mostrado em função de uma série de parâmetros (como por exemplo a razão do número de eventos executados corretamente e o número de eventos em relação ao número total de eventos executados, ao número de *rollbacks*, ao tamanho médio do *rollback* e ao número de antimensagens enviadas). Cada processo lógico possui seu próprio valor da janela, que aumenta ou diminui periodicamente, dependendo da mudança do trabalho útil executado (Das 1996).

3.2.18 Protocolo Probabilístico Baseado em Custos

É similar ao *Adaptive Time Warp* (seção 3.2.8) e tem como base a análise da probabilidade de ocorrência de *rollbacks*. Cada processo lógico monitora a marca de tempo de cada mensagem que chega, assim como o comportamento de *rollback*, para definir os limites de uma janela de tempo para bloquear o mesmo (Das 1996).

3.2.19 *Local Adaptive Protocol* (*LAP*)

Esse protocolo tenta estimar uma *blocking window* de tempo real, com base em incrementos médios do *LVT* e tempos médios entre-chegada observados nos canais de entrada, e utiliza mensagens nulas para prevenir *deadlocks* (Ferscha 1995a). Um processo lógico bloqueia se estima que um evento chegando em um de seus canais de entrada vazios irá causar *rollback* (Das 1996).

3.2.20 *Composite ELSA*

Nesse protocolo o sistema simulado é modelado como um grafo dirigido onde os vértices representam os processos lógicos e os arcos a interconexão entre os nós (Arvind; Smart 1992). Um processo lógico pode chavear entre modo conservativo ou modo otimista quando determina que um evento é seguro ou não (Das 1996).

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

Esse protocolo faz com que um processo lógico não selecione sempre a primeira mensagem da lista de eventos. Se alguma condição não é satisfeita o processo lógico 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).

3.2.22 *Tentative Time Warp (TTW)*

Esse protocolo altera o mecanismo de controle local do *Time Warp* de forma que as mensagens sejam canceladas em grupo ao invés de forma isolada. O mecanismo de controle global permanece como no *Time Warp* (Kalantery 1997).

3.2.23 *Adaptive Throttle Scheme*

Baseado no fato de que o desempenho do *Time Warp* depende muito do progresso de cada processo lógico no tempo, esse protocolo propõe o uso da *Global Progress Window* (*GPW*) para indicar como progridem no tempo o processo lógico mais lento e o mais rápido. Esse esquema ajusta a velocidade de execução dos eventos no *Time Warp*, permitindo aos processos lógicos acelerar ou suspender sua execução (Tay et al. 1997).

3.2.24 *Length-based Blocking*

Nesse protocolo um processo lógico permanece bloqueado (para execução de eventos) até que o comprimento de sua fila de entrada ultrapasse um valor limite. Como motivação, o autor ressalta que em muitos casos a probabilidade de encontrar uma violação de causa e efeito diminui conforme o número de eventos na fila aumenta (atrasando a execução dos eventos, uma mensagem que poderia provocar *rollback* no *Time Warp* básico pode ser recebida e inserida na fila de entrada do processo lógico sem causar nenhum dano à simulação). Conseqüentemente, o bloqueio é uma maneira de diminuir a probabilidade de *rollbacks* permitindo que mais eventos sejam inseridos na fila de entrada antes de qualquer execução (Pham; Fdida 1996).

3.3 Classificações para Protocolos Otimistas

As pesquisas na área de simulação distribuída têm convergido para variações híbridas de protocolos de sincronização que implementam uma abordagem intermediária entre os extremos conservativo e otimista, uma vez que a escolha por um determinado protocolo é uma tarefa bastante complexa. Esses protocolos

utilizam uma forma de otimismo controlado, aproveitando os benefícios da visão otimista (explorar mais paralelismo) sem serem afetados pelos seus problemas (sobrecarga causado por *rollbacks* e consumo de memória) (Reiher; Jefferson 1989), ou então adicionam mais otimismo em alguma abordagem conservativa.

Na literatura de simulação distribuída 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 forma mais organizada (Spolon et al. 2000b). A construção de uma taxonomia concisa e hierárquica pode facilitar as análises comparativas de desempenho entre os protocolos (com a idéia de *plugins* que será apresentada no capítulo 4). O uso da taxonomia pode identificar os aspectos mais importantes do protocolo *Time Warp* e que merecem atenção em um estudo de desempenho através da técnica de modelagem (capítulo 4).

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, considerando tanto os protocolos que introduzem otimismo em protocolos conservativos quanto os protocolos que introduzem idéias conservativas em protolocos otimistas. Segundo o autor, alguns protocolos são difíceis de se classificar.

Srinivasan apresenta uma classificação e define uma classe de protocolos. A primeira aborda os protocolos que limitam o otimismo no *Time Warp* (com base no critério que os mesmos utilizam) (Srinivasan; Reynolds 1995a) e a segunda define uma classe de protocolos adaptativos, isto é, que ajustam seu comportamento dinamicamente, de acordo com as variações na simulação (Srinivasan; Reynolds 1995b).

As seções a seguir apresentam as classificações descritas anteriormente.

3.3.1 Classificação de Das

A classificação apresentada por Das (Das 1996) divide os protocolos em Híbridos ou Adaptativos, como descrito na figura 3.1.

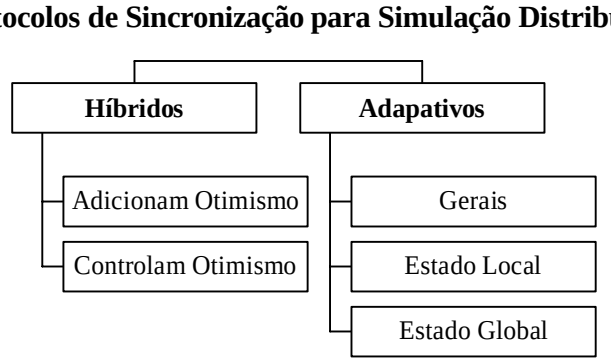


Figura 3.1: Classificação de Das.

Os protocolos Híbridos abrangem os protocolos que adicionam otimismo a protocolos conservativos, como por exemplo Simulação Especulativa, *SRADS*, *Breathing Time Buckets*, *Filtered Rollback*, e os protocolos que buscam controlar o otimismo no *Time Warp,* como por exemplo *Moving Time Window*, *Bounded Time Warp, MIMDIX*, *Breathing Time Warp, Composite ELSA*, *Local Time Warp* e *Filter*.

Segundo o autor, os protocolos Adaptativos ajustam-se entre a abordagem otimista e a abordagem conservativa com o objetivo de minimizar o tempo de execução da simulação distribuída. São subdivididos em Gerais, Estado Local e Estado Global.

Os protocolos Gerais têm como exemplo *Moving Time Windows* e *MIMDIX*. Os protocolos com base em informações referentes ao Estado Local do processo lógico englobam *Penalty-based Throttling, Adaptive Time Warp, Protocolo Probabilístico baseado em Custos, CO-OP, Probabilistic Distributed Discrete Event Simulation Protocol, Local Adaptive Protocol, PADOC* e *Adaptive Bounded Time Window*. Os protocolos que utilizam informações referentes ao Estado Global da simulação distribuída agrupam *Adaptive Memory Management Protocol*, protocolos *Near Perfect State Information* (*NPSI*) e *Cancelback Protocol.*

3.3.2 Classificação de Srinivasan

A classificação e a classe de protocolos propostos por Srinivasan foram publicadas em trabalhos distintos e abordam de forma independente os protocolos que limitam o otimismo no *Time Warp* (Srinivasan; Reynolds 1995a) e os protocolos que se adaptam às mudanças no estado da simulação (Srinivasan; Reynolds 1995b). A figura 3.2 descreve essa classificação.

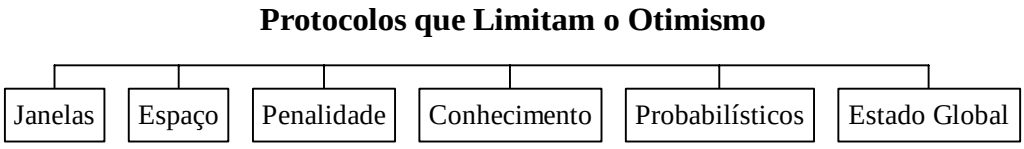


Figura 3.2: Classificação de Srinivasan.

Segundo Srinivasan os protocolos podem ser divididos em seis grupos: Janelas, Espaço, Penalidade, Conhecimento, Probabilísticos e Estado Global.

Os protocolos com base em Janelas permitem que somente os eventos com valor de marca de tempo dentro do intervalo da janela de tempo sejam executados. Como exemplo pode-se citar *Moving Time Windows*, *Window based Throttling*, *Bounded Time Warp* e *Breathing Time Warp*.

Os protocolos baseados no Espaço utilizam limites espaciais ao invés de limites temporais (janelas) para limitar o otimismo. Os processos lógicos são divididos em grupos, e *Time Warp* é utilizado para efetuar o sincronismo dentro de cada grupo. Os grupos interagem sem risco, de forma conservativa. Pode-se citar como exemplo *Local Time Warp* e *SRADS*.

De acordo com o autor, nos protocolos baseados em Penalidade utiliza-se o comportamento da simulação para penalizar (bloquear) processos lógicos enquanto outros são favorecidos (podem continuar a execução). Como exemplo tem-se *Penalty-based Throttling*.

Os protocolos com base em Conhecimento utilizam informação sobre a ocorrência de *rollbacks* para restringir a propagação de computação provavelmente incorreta. Exemplos incluem *Filter* e *WOLF*.

Os protocolos Probabilísticos fazem uma previsão probabilística sobre o comportamento dos processos lógicos. Como exemplo tem-se *MIMDIX*.

Finalizando, os protocolos com base no Estado Global da simulação analisam a simulação distribuída como um todo para limitar o excesso de otimismo do *Time Warp.* Exemplos são *Adaptive Memory Management* e os protocolos *NPSI.*

Srinivasan (Srinivasan; Reynolds 1995b) também propõe que o controle do otimismo no *Time Warp* seja classificado, devido ao fato de que alguns protocolos introduzem atrasos entre as execuções dos eventos. Esses protocolos foram denominados *Asynchronous Adaptive Waiting Protocols* (*AAWP*). Como exemplos pode-se citar *Penalty-based Throttling*, *Adaptive Time Warp*, *Local Adaptive Protocol*, protocolos *NPSI, Breathing Time Warp* e *UDS.*

3.4 Análise Comparativa

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 dos mesmos. A tabela 3.1 apresenta todos os protocolos estudados, o ano de publicação, o tipo de método utilizado na sincronização, os protocolos dos quais deriva alguma característica e a classificação (Das ou Srinivasan) à qual pertence. As demais tabelas dividem os protocolos em grandes grupos, conforme as classificações apresentadas: janelas (tabela 3.2), probabilísticos (tabela 3.3), conhecimento (tabela 3.4), estado (tabela 3.5), classificados de acordo com Das e Snirivasan.

Tabela 3.1: Características dos Protocolos que Limitam Otimismo no *Time Warp*.

Protocolo	Tipo de Sincronização	Relacionado com	Classificação
Wolf (1988)	Quando ocorre *rollback* faz difusão para encerrar rapidamente a computação errônea	*Time Warp*	Das, Srinivasan
Penalty-based Throttling (1989)	Penaliza os processos lógicos que efetuam muito *rollback*	*Time Warp*	Das, Srinivasan
Window-based Throttling (1989)	Janelas	*Time Warp*	Srinivasan
Moving Time Windows (1989)	Janelas	*Time Warp*	Das, Srinivasan
UDS (1990)	Janelas	*Time Warp*	Srinivasan
Adaptive Time Warp (1990)	Janelas adaptativas	*Time Warp*	Das, Srinivasan
SRADS (1990)	Usa a idéia de simulação especulativa, porém as filas de	Simulação Especulativa	Srinivasan

	eventos e os estados podem ser alterados, não usa antimensagens		
Breathing Time Buckets (1991)	Ciclos determinados pelo horizonte de eventos; como não utiliza antimensagens, não apresenta sobrecarga para manipulá-las, como no *Time Warp*. Para que o mecanismo seja eficiente, exige que em um ciclo sejam processados eventos suficientes; necessita de técnicas sofisticadas para determinar quando o último processo lógico ultrapassou o horizonte de eventos	*Time Buckets e Time Warp*	Nenhuma
Filter (1991)	Cada mensagem carrega uma lista de supostas classes de mensagens que poderiam levar ao seu cancelamento	*Time Warp*	Das, Srinivasan
Simulação Especulativa (1991)	Executa os eventos em elementos de processamento que estão desocupados, sem alterar o estado e a fila de eventos; faz análise dos dados para verificar se está correto	*Time Warp*	Nenhuma
Conservative-Optimistic Protocol (CO-OP) (1991)	Processa os eventos em termos de sua probabilidade de estarem corretos, não estritamente por ordem de tempo	*Time Warp*, pois utiliza *rollbacks*	Das
Sfilter (1992)	Cada mensagem carrega suposições locais	*Filter*	Nenhuma
Composite ELSA (1992)	O sistema simulado é visto como um grafo dirigido; cada nó pode chavear entre otimista e conservativo	otimista e conservativo	Das
Breathing Time Warp (1992)	Horizonte de eventos e processamento otimista	*Time Warp, Breathing Time Buckets*	Das
Bounded Time Warp (1992)	Janelas	*Time Warp*	Das, Srinivasan
MIMDIX (1992)	Sincroniza os processos lógicos em intervalos δ determinados probabilisticamente	*Time Warp*	Das, Srinivasan
Local Time Warp (1993)	Sincronização otimista dentro dos grupos de processos lógicos e conservativa entre os grupos	*Time Warp e Conservative Time Windows*	Das, Srinivasan
Adaptive Bounded Time Window (1993)	Cada processo lógico possui seu próprio valor da janela, que aumenta ou diminui periodicamente, dependendo da mudança do trabalho útil executado	*Time Warp*	Das
Aggressive Windowing (1993)	Janelas (conservativa e agressiva), antimensagens, cancelamento agressivo e *rollback*	*Time Warp*	Nenhuma
Local Adaptive Protocol (1994)	Tenta estimar uma *blocking window* de tempo real	*CMB*, usa mensagens nulas para prevenir *deadlock*	Das, Srinivasan
Probabilistic DDES Protocol (1994)	Utiliza janelas cujos tamanhos são variáveis e calculados probabilisticamente	*Time Warp*	Das

PADOC (1995)	Bloqueia temporariamente o processamento dos eventos, de forma probabilística	*CMB* e *Time Warp*	Das
Algoritmos Adaptativos *NPSI* (1995)	Agressividade e o risco são controlados dinamicamente através de informação do estado da simulação	*Time Warp*	Das, Srinivasan (*AAWP*)
Protocolo Probabilístico Baseado em Custos (1995)	Calcula probabilisticamente uma largura ótima para uma *blocking window*	*ATW*	Das
Length-based Blocking (1996)	Bloqueia processos lógicos até que a fila de entrada atinja um valor limite	*Time Warp*	Nenhuma
Adaptive Throttle Scheme (1997)	Ajusta a velocidade de execução dos processos lógicos	*Time Warp*	Nenhuma
Tentative Time Warp (1997)	Mecanismo de controle local baseado em cancelamento de grupos; o controle global permanece como no *Time Warp*	*Time Warp*	Nenhuma
State Query Time Warp (1997)	Os processo lógicos armazenam informações que permitem "validar" um evento para execução	*Time Warp*	Nenhuma

A tabela 3.2 apresenta uma análise resumida das características dos protocolos baseados em janelas e citados nas seções anteriores. De uma forma geral, tamanhos de janelas muito pequenos podem limitar a ocorrência de *rollbacks* porém exploram menos o paralelismo, enquanto tamanhos de janelas grandes podem implicar na situação contrária e permitir a execução de eventos fora de ordem cronológica. O cálculo e o tempo gastos para calcular o tamanho da janela não podem ser complexos demais a ponto de ser mais custoso utilizá-las do que deixar os *rollbacks* acontecerem.

Tabela 3.2: Comparação dos Protocolos de Janelas.

Protocolo	**Análise**
Window-based Throttling	Os processos lógicos fora da janela são bloqueados; tamanho da janela muito pequeno degrada o desempenho, e muito grande perde o sentido
Moving Time Windows	As janelas têm tamanho pré-definido; problema: como determinar o valor de Δ.
UDS	Os processos lógicos são bloqueados por períodos de janelas
Adaptive Time Warp	Bloqueia os processos lógicos por um período denominado *blocking window* (janelas adaptativas)
Breathing Time Warp	Os primeiros N eventos são processados como no *Time Warp*, depois utiliza *Breathing Time Buckets.* Ao liberar somente as mensagens de N_1 eventos além de *GVT*, evita avalanches de antimensagens; o usuário deve determinar o valor de N.
Bounded Time Warp	Para determinar quando todos os processos lógicos estão inativos, utiliza um *token* que circula entre eles. Problema: como determinar o

	limite (*bound*)
Adaptive Bounded Time Window	As janelas são calculadas dinamicamente, de acordo com o trabalho útil executado pelos processos lógicos
Adaptive Throttle Scheme	Ajusta a velocidade de execução dos eventos no *Time Warp*, permitindo aos processos lógicos acelerar ou suspender sua execução
Aggressive Windowing	Problema: como determinar o tamanho da janela

Tabela 3.3: Comparação dos Protocolos Probabilísticos.

Protocolo	Análise
Conservative-Optimistic Protocol (CO-OP)	Esse protocolo processa os eventos em termos de sua probabilidade de estar correto, não estritamente por ordem de tempo (menor marca de tempo primeiro), porém a tarefa de como verificar a probabilidade de estar correto é complexa
MIMDIX	Em intervalos determinados probabilisticamente, os processos lógicos são sincronizados por um processo especial porém a aplicação pode escolher o valor de δ e as probabilidades para sincronização e os processos lógicos podem desfazer trabalho correto; a existência de um elemento centralizado para cuidar da sincronização pode ser um gargalo; se o intervalo de re-sincronização é pequeno, o protocolo torna-se muito conservativo e se é grande, a re-sincronização deixa de ter sentido (Rajaei et al. 1993);
Probabilistic Distributed Discrete Event Simulation Protocol	Calcula uma probabilidade em função das mensagens que chegam em cada processo lógico
Protocolo Probabilístico Baseado em Custos	Utiliza uma função probabilística envolvendo a probabilidade e o custo de *rollbacks* para determinar uma *blocking window*, ou seja, um período para o processo lógico ficar bloqueado
PADOC	Cada processo lógico monitora as mensagens que chegam e, pelo padrão de chegada, conseguem se adaptar à próxima mensagem, de acordo com abordagem otimista ou conservativa

Tabela 3.4: Comparação dos Protocolos Baseados em Conhecimento.

Protocolo	Análise
Filter	Cada mensagem carrega uma lista de suposições sobre o que pode causar *rollback*; o cancelamento de eventos é feito rapidamente, através de uma única mensagem (Prakash; Subramanian 1992); a lista de suposições pode aumentar a sobrecarga por mensagem, mas pode também reduzir o número total de mensagens enviadas, o número de mensagens processadas e o número de *rollbacks*
Sfilter	Como *Filter*, porém não necessita guardar as mensagens enviadas
WOLF	Faz difusão para encerrar a computação errônea

Tabela 3.5: Comparação dos Protocolos Baseados no Estado.

Protocolo	Análise
Algoritmos Adaptativos *NPSI*	Utilizam informações baseadas no estado local para controlar o otimismo
Local Adaptive Protocol	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, porém utiliza mensagens nulas, podendo levar a sobrecarga de comunicação
Penalty-based Throttling	Penaliza os processos lógicos que executam mais *rollbacks*
Length-based Blocking	Um processo lógico permanece bloqueado até que sua fila de entrada atinja um tamanho limite
Composite ELSA	Um processo lógico pode chavear entre modo conservativo ou modo otimista quando determina que um evento é seguro ou não

3.5 Proposta de uma Taxonomia Hierárquica

A área de simulação distribuída é carente em relação a classificações abrangentes dos protocolos de sincronização. Para preencher essa lacuna, este trabalho sugere uma extensão para as classificações dos protocolos otimistas apresentadas, buscando agrupar de forma natural a grande variedade de protocolos propostos ao longo dos anos. As classificações apresentadas nas seções 3.3.1 e 3.3.2 foram utilizadas como base para produzir uma taxonomia mais geral e que procura ser mais simples e completa. Com isso, a tarefa de estudar e comparar os vários mecanismos de sincronização baseados no *Time Warp* torna-se mais simples. Essa classificação consiste em uma atualização da classificação proposta em (Spolon et al. 1999) e permite uma melhor visualização dos mecanismos otimistas, consistindo em uma abordagem genérica e hierárquica.

Além da necessidade de se agrupar melhor os protocolos que limitam o *Time Warp*, com a proposta da nova taxonomia procura-se corrigir as falhas apresentadas por Das e Srinivasan. Alguns protocolos desenvolvidos posteriormente à divulgação das duas classificações não foram considerados, como é o caso dos protocolos *Tentative Time Warp*, *State Query Time Warp*, *Length-based Blocking* e *Adaptive Throttle Scheme*.

Além disso, nenhuma das classificações apresentadas considera o fato de que existem diferentes algoritmos que podem ser seguidos para efetuar determinadas tarefas, como por exemplo o salvamento de estados e o cancelamento de mensagens. Esses e outros algoritmos, como mostrado na literatura, podem influenciar no

desempenho do protocolo e, por isso, serão considerados na proposta de taxonomia apresentada nesta seção.

A classificação de Das apresenta-se inconsistente ao separar os Protocolos que Controlam Otimismo dos Protocolos Adaptativos. Ambos procuram limitar o otimismo do *Time Warp* podendo executar este trabalho de forma estática (não adaptativa) ou dinâmica (adaptativa), que Das não menciona. Além disso, a noção de protocolo Híbrido abrange também aqueles que adicionam otimismo a protocolos conservativos, ou seja, protocolos cujas características principais são da abordagem *CMB*. Esses protocolos, por apresentarem características intrínsecas de abordagens conservativas, como por exemplo o uso de mensagens nulas, não serão considerados na taxonomia proposta.

Das também classifica *Local Adaptive Protocol* como um protocolo que se utiliza do estado da simulação para limitar o otimismo. Apesar de cumprir esse papel, esse protocolo apresenta *deadlocks* e mensagens nulas, como no *CMB*, e também não será considerado na taxonomia proposta.

A classificação de Srinivasan cria grupos de protocolos com base em Penalidade e Estado Global e não considera protocolos que fazem uso de informações referentes ao estado local do processo lógico. A característica básica de todos esses protocolos de sincronização é a utilização de informações relativas ao estado da simulação para definir se os processos lógicos podem ou não executar eventos, ou seja, têm como base o Estado da simulação. O grupo dos protocolos de Conhecimento também se utiliza de informações relativas ao estado da simulação distribuída, porém essa informação é difundida para outros processos lógicos, então a denominação mais adequada é Difusão. Srinivasan também não separa os protocolos adaptativos dos não adaptativos.

A classificação de Das insere o protocolo *Adaptive Time Warp* entre aqueles que têm como base o estado local do processo lógico, porém a característica marcante deste protocolo é o fato de que o processo lógico define uma janela de tempo para que o mesmo fique bloqueado.

A classificação de Das considera que os protocolos *CO-OP*, *Probabilistic Distributed Discrete Event Simulation Protocol*, *PADOC* e *Adaptive Bounded Time Window* têm como base o estado da simulação. Porém, o protocolo *Adaptive Bounded Time Window* apresenta a característica principal de utilizar janelas de

tempo e os demais fazem uso da probabilidade para decidir se executam ou não um evento.

Além disso, ambas as classificações tratam o protocolo *Adaptive Memory Management* como sendo um protocolo baseado no estado global da simulação. É correto afirmar que esse protocolo, e também o protocolo *Cancelback*, limitam o otimismo do *Time Warp* observando o estado da simulação, porém essa limitação é feita de maneira indireta. O objetivo principal consiste em efetuar o gerenciamento da memória disponível para salvar estados, mas ao imporem aos processos lógicos um espaço limitado na memória disponível estão também prevenindo que os mesmos avancem muito no tempo de simulação. Isso justifica a criação de uma nova classe na taxonomia, como explicado nas seções seguintes.

A classificação para os mecanismos otimistas foi desenvolvida em uma estrutura concisa e hierárquica que fornece uma visão global dos mecanismos e permite o seu estudo em grupos, ao invés de analisá-los isoladamente. A taxonomia proposta apresenta três pontos de vista: de acordo com as Características de Implementação do mecanismo de sincronização (figura 3.3(a)), de acordo com as Características do Mecanismo de Sincronização propriamente dito (Spolon et al. 1999) (figura 3.3(b)) e de acordo com o Modo de Limitar o Otimismo (figura 3.3(c)).

A taxonomia proposta considera apenas os protocolos que têm como base a limitação do otimismo no *Time Warp*, o que exclui protocolos como, por exemplo, *Local Adaptive Protocol*, que está incluído nas classificações de Das e Srinavasan, e os protocolos que adicionam otimismo a protocolos conservativos classificados por Das.

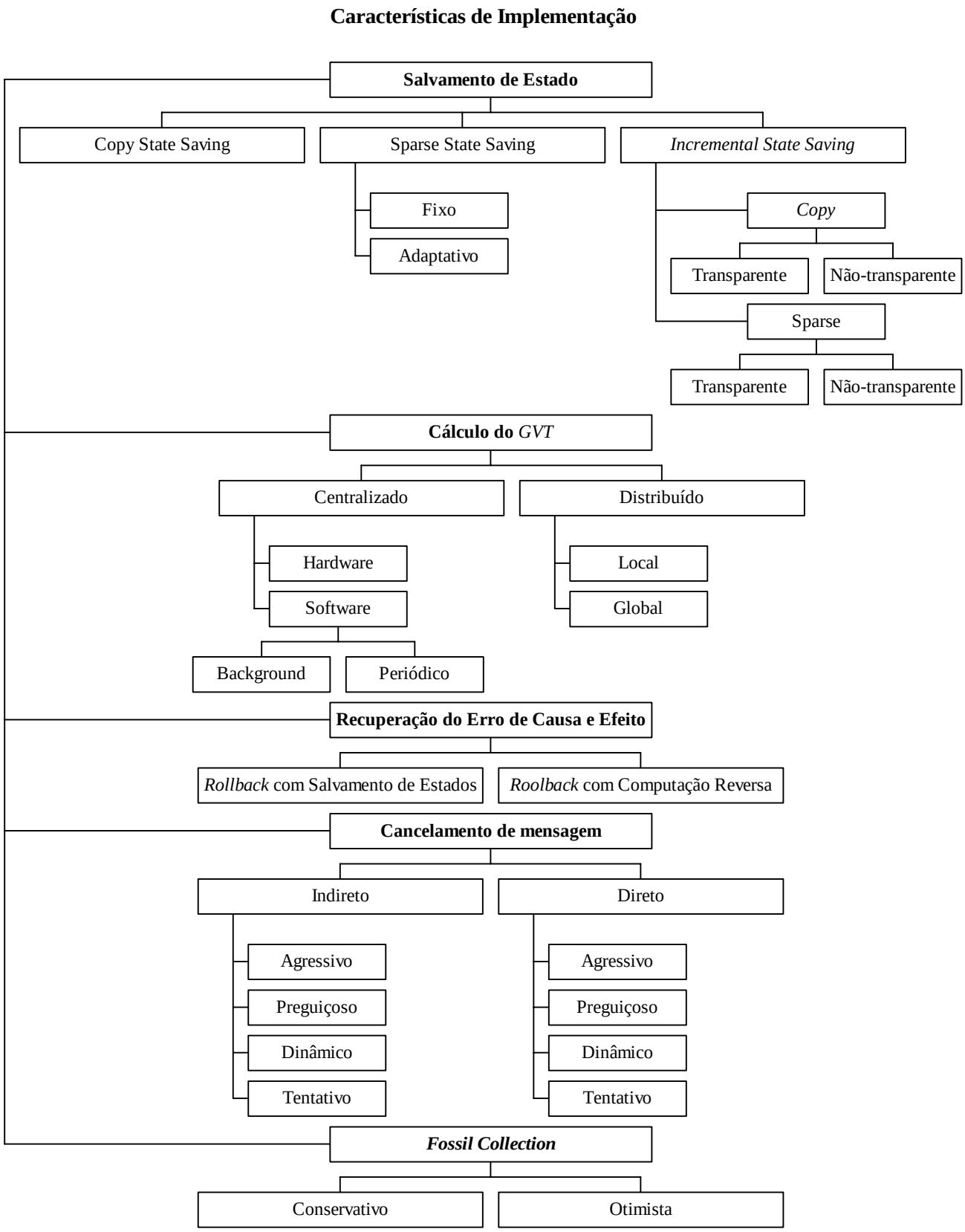


Figura 3.3(a): Taxonomia para Protocolos que Limitam Otimismo no *Time Warp* (Características de Implementação).

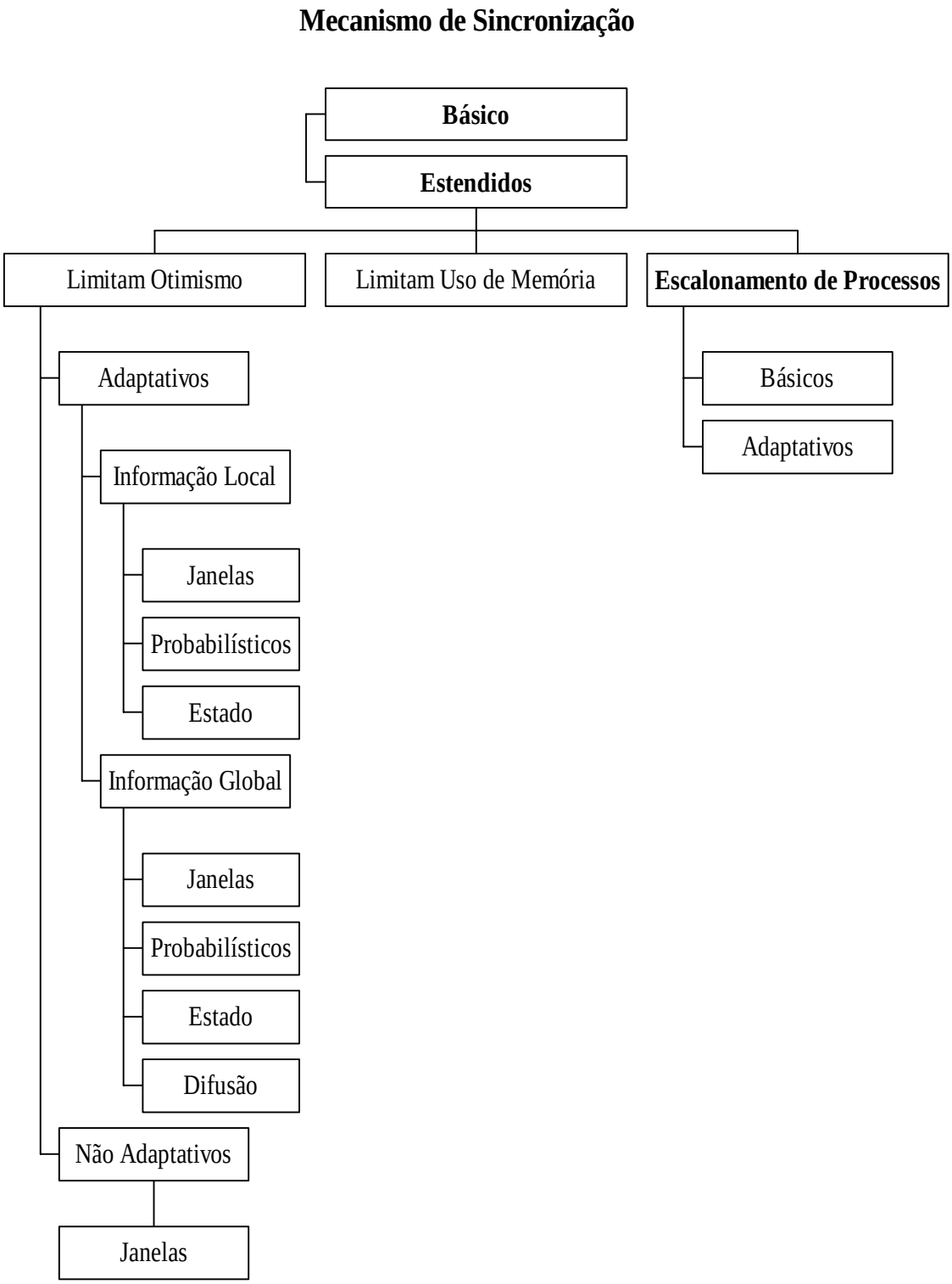


Figura 3.3(b): Taxonomia para Protocolos que Limitam Otimismo no *Time Warp* (Mecanismo de Sincronização).

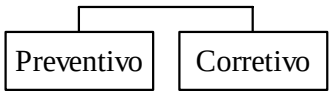


Figura 3.3(c): Taxonomia para Protocolos que Limitam Otimismo no *Time Warp* (Modo de Limitar o Otimismo).

A classificação que considera as características de Implementação é discutida na seção 3.5.1, estando justificada pelo fato de que não existe um consenso na área de simulação distribuída com relação à melhor maneira de se implementar protocolos baseados no *Time Warp* (Carothers et al. 2000).

A classificação com base no Mecanismo de Sincronização apresenta os protocolos divididos em duas categorias, Protocolo Básico e Protocolos Estendidos. O primeiro corresponde ao mecanismo proposto por Jefferson (Jefferson 1985) enquanto os Protocolos Estendidos correspondem a variantes do *Time Warp* que procuram reduzir o excesso de otimismo da idéia original de Jefferson. A seção 3.5.2 apresenta essa classificação.

A classificação com base no Modo de Limitar o Otimismo justifica-se pelo fato de que alguns protocolos limitam o otimismo sem fazer análise do estado da simulação ou do processo lógico (por exemplo, com respeito ao número de *rollbacks* e eficiência), mas adaptam-se durante a execução. A seção 3.5.3 discute essa abordagem.

3.5.1 Classificação Segundo as Características de Implementação

A proposta original do *Time Warp* define a noção de tempo virtual e fornece as diretrizes de como efetuar a sincronização entre os processos lógicos usando a abordagem otimista com *rollbacks* e salvamento de estados. Porém, no trabalho de Jefferson não estão especificadas, para citar alguns exemplos, a forma pela qual as antimensagens devem ser enviadas, a maneira como o *rollback* deve ser efetuado ou ainda como e quando salvar as informações relativas aos estados dos processos lógicos e como efetuar o cálculo do *GVT*.

Isso implica que os projetistas e pesquisadores contam com uma gama variada de opções, além de poderem criar suas próprias soluções no desenvolvimento de seus ambientes de simulação que fornecem suporte ao *Time Warp*.

As seções a seguir descrevem cada uma das classes apresentadas na taxonomia da figura 3.3(b).

3.5.1.1 Classificação por Salvamento de Estados

O *Time Warp* é um protocolo de sincronização otimista que permite que a simulação distribuída execute até que ocorra um erro de causa e efeito. Quando isso ocorre, *Time Warp* executa um *rollback* para retornar a simulação para um estado consistente. A execução do *rollback* juntamente com o retorno da simulação a um estado consistente exige alguns procedimentos que podem afetar o desempenho da simulação. O *Time Warp* necessita de algoritmos para decidir quando e como executar o salvamento de estados, descritos a seguir.

O procedimento de salvamento de estados no *Time Warp* pode ser executado por um dos seguintes algoritmos, que definem três abordagens distintas (figura 3.4):

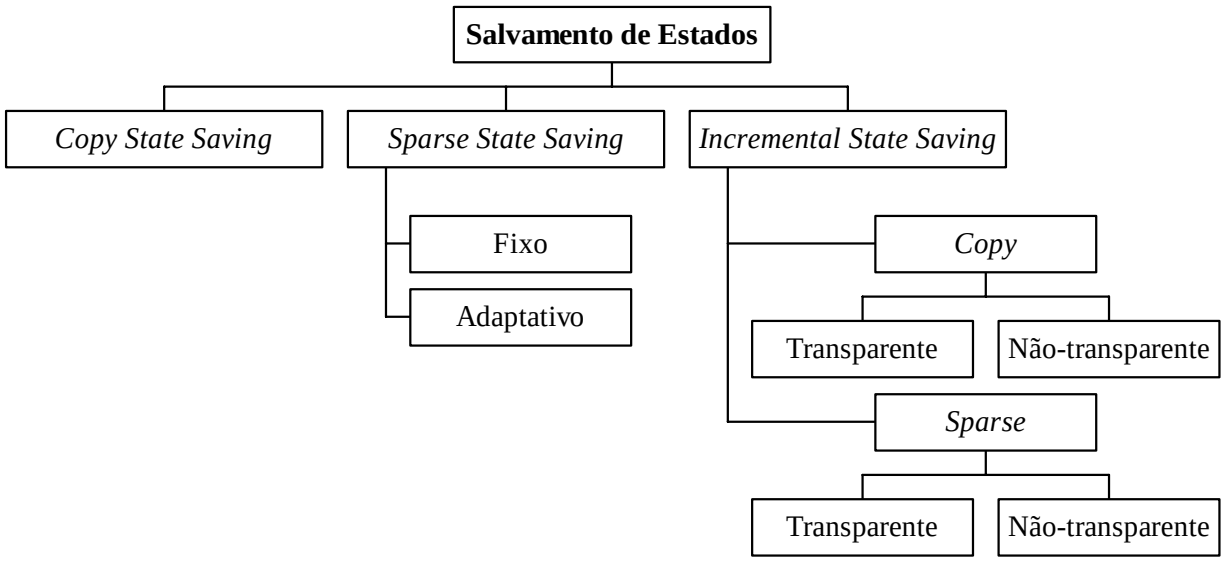


Figura 3.4: Classificação por Salvamento de Estados.

- *Copy state saving*: todo o estado do processo lógico é salvo após a execução de cada evento. É a abordagem mais simples;

- *Sparse state saving*: todo o estado do processo lógico é salvo periodicamente e qualquer estado pode ser recuperado através da restauração do último estado salvo antes do evento afetado pelo *rollback* e pela re-execução de eventos intermediários. Pode ser fixo (o intervalo de tempo entre os salvamentos de estado é constante durante toda a execução da simulação) ou adaptativo (o intervalo de tempo entre os salvamentos de estado é escolhido dinamicamente, de acordo com o comportamento da simulação). Alguns métodos descritos na literatura são o Método de Rönngren Baseado no Tempo de Execução, o Método de Rönngren Baseado no Consumo de Memória e o Método de Fleischmann-Wilsey (mais detalhes podem ser encontrados em (Quaglia 1998, Quaglia 1999, Rönngren et al. 1996a, Soliman 1999));
- *Incremental state saving*: somente as partes do estado do processo lógico que foram modificadas são salvas. Pode ser dividido em transparente ou não transparente. No caso não-transparente, todas as variáveis de estado que foram modificadas devem ser identificadas pelo usuário da simulação e atualizadas (ou seja, é tarefa do usuário identificar o que mudou e chamar explicitamente um procedimento ou rotina para salvar essa parte alterada), e no caso transparente, as variáveis de estado que sofreram modificação são atualizadas automaticamente pelo sistema de simulação (Rönngren et al. 1996b, Rönngren et al. 1996c). A implementação pode optar pelo caso *Copy* e salvar as partes modificadas do estado a cada alteração, ou então determinar intervalos de tempo para efetuar o salvamento, identificando-se ao caso *Sparse*. A biblioteca de simulação distribuída *Compose* (*Conservative Optimistic and Mixed Parallel-oriented Simulation Environment*) é um exemplo de utilização de *Incremental state saving* (Meyer et al. 2000).

3.5.1.2 Classificação por Cálculo do *GVT*

O grande problema associado à tarefa de salvar estados consiste na utilização do espaço de memória disponível. Dessa forma, outro ponto importante na implementação de um sistema *Time Warp* reside na adoção de alguma técnica para

descartar os estados que foram salvos mas que não serão mais necessários. O valor do *GVT* garante que os estados salvos com marcas de tempo menores podem ser descartados, pois não ocorre *rollback* para estados anteriores ao *GVT*.

Outra razão importante para efetuar o cálculo do *GVT* em uma simulação distribuída consiste em oferecer suporte para a simulação distribuída interativa (*DIS – Distributed Interactive Simulation*). Somente quando os eventos executados na simulação são seguros, ou seja, suas marcas de tempo são menores do que o *GVT* e portanto não irão sofrer *rollback*, é que é permitido que tais eventos liberem informação para o mundo externo. Isso ocorre em jogos ou então em simulações para treinamento militar (Steinman et al. 1995).

O fator mais importante e que torna o cálculo do *GTV* uma tarefa difícil é o fato de existirem mensagens/antimensagens circulando pela rede de comunicação. Essas mensagens/antimensagens devem ser consideradas no momento de se determinar qual a menor marca de tempo da simulação distribuída (Hagenauer 1999).

Existem diferentes abordagens para o cálculo do valor do *GVT* (figura 3.5). Pode-se dividir essas abordagens em centralizadas ou distribuídas. Na abordagem centralizada o valor do *GVT* é calculado por um processo lógico (software) ou hardware específico (D'Souza et al. 1997). A vantagem de se ter um hardware específico para efetuar tal tarefa é que existe pouca influência no desempenho da simulação, porém tem-se um aumento no custo e na complexidade.

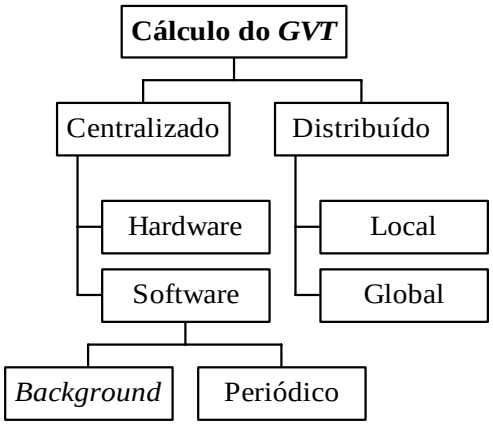


Figura 3.5: Classificação por Cálculo do *GVT*.

A abordagem de um processo lógico específico (software) é a mais utilizada na literatura, e nesse caso tem-se o problema de minimizar o impacto do algoritmo de

cálculo no desempenho da simulação distribuída. Esse algoritmo pode ser executado em *background* (concorrendo assim com os demais processos lógicos) ou pode ser iniciado periodicamente (por exemplo, quando um processo lógico necessita efetuar *fossil collection* para poder continuar a execução dos eventos). Nesse caso os processos lógicos devem esperar o novo valor do *GVT* (Steinman et al. 1995).

Uma vantagem da abordagem distribuída para o cálculo do *GVT* é a ausência de sobrecarga em um único processo lógico, pois o trabalho é distribuído por todos. Essa abordagem também pode ser aplicada de forma local, onde os processos lógicos vizinhos calculam seu *GVT* que depois é comparado aos demais *GVTs* das outras vizinhanças e o menor é eleito o *GVT*, ou então de forma global, onde um único valor de *GVT* é calculado considerando-se todos os processos lógicos. *Speed GVT* (Steinman et al. 1995) é um exemplo de abordagem distribuída global.

3.5.1.3 Classificação com Base na Recuperação de Erro de Causa e Efeito

Os protocolos de sincronização otimistas, como citado no capítulo 2, caracterizam-se pelo fato de que os processos lógicos podem executar os eventos conforme eles são recebidos, sem ter a certeza de que a ordem dos eventos recebidos é realmente correta e não irá acarretar problemas para a simulação.

Porém, quando ocorre um erro de causa e efeito, o protocolo de sincronização precisa corrigí-lo e reiniciar a simulação a partir de um ponto seguro no tempo. Existem duas formas de se corrigir um erro de causa e efeito (figura 3.6). A maneira mais comum é a proposta inicial de Jefferson (Jefferson 1985). Nessa técnica os valores dos estados pelos quais o processo lógico passa são salvos e a simulação pode ser restaurada para um estado seguro através do *rollback*, que copia de volta os valores do estado salvo para as variáveis da simulação. *GTW* (Fujimoto 2000) e *Warped* (Radhakrishnan et al. 1998) são exemplos de sistemas de simulação distribuída *Time Warp* que utilizam o *rollback* e salvamento de estados para recuperar erros de causa e efeito.

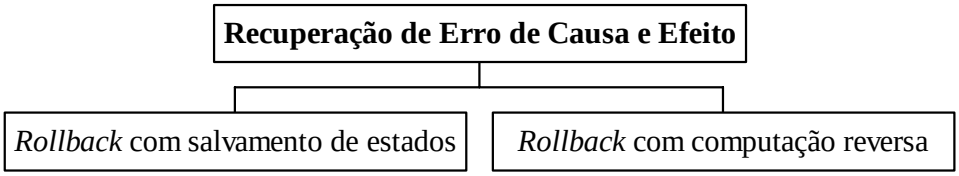


Figura 3.6: Classificação por Recuperação do Erro de Causa e Efeito.

Outra forma de se recuperar um erro de causa e efeito consiste em implementar Computação Reversa, proposta por Carothers (Carothers et al. 1999a, Carothers et al. 1999b). Como exemplo tem-se o sistema *ROSS* (*Rensselaer's Optimistic Simulation System*) (Carothers et al. 2000). Nessa técnica, o *rollback* é realizado pela execução das operações inversas relativas às operações individuais que são efetuadas durante a execução de um evento. O sistema garante que as operações inversas restauram o estado da simulação para os valores que antecedem a execução do evento. A vantagem dessa técnica reside na pequena quantidade de informação de controle que precisa ser salva, em relação ao tamanho de todo o estado (Carothers et al. 1999b).

3.5.1.4 Classificação por Cancelamento de Mensagens

Outro fator importante que deve ser levado em consideração nas implementações baseadas no *Time Warp* relaciona-se às mensagens que são enviadas de forma errada quando ocorre um erro de causa e efeito. Essas mensagens devem ser canceladas quando um *rollback* é executado, através do envio explícito de antimensagens (Cancelamento Indireto) ou do cancelamento das mensagens enviadas nas filas dos outros processos lógicos (Cancelamento Direto). Nesse caso, quando a execução de um evento resulta no envio de um novo evento para um processo lógico remoto, um apontador para este novo evento é mantido juntamente com o evento executado, na estrutura de dados do processo emissor. Isso elimina a necessidade do envio explícito de antimensagens (maiores detalhes podem ser encontrados em Carothers et al. 1999b, Reiher et al. 1990).

Em ambos casos, as abordagens utilizadas são Cancelamento Agressivo, Cancelamento Preguiçoso, Cancelamento Dinâmico e Cancelamento por Tentativa (figura 3.7).

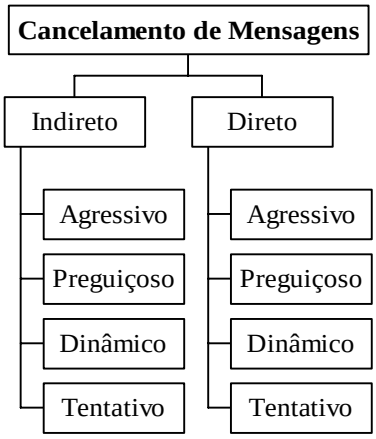


Figura 3.7: Classificação por Cancelamento de Mensagens.

Quando Cancelamento Agressivo é adotado e um processo lógico sofre um *rollback* para o tempo *t*, são enviadas antimensagens imediatamente para anular todas as mensagens enviadas com marca de tempo maior do que *t*.

No Cancelamento Preguiçoso o processo lógico espera para ver se a reexecução dos eventos executados fora de ordem produz as mesmas mensagens. Em caso afirmativo não é necessário enviar antimensagens e no caso contrário as antimensagens são enviadas.

O Cancelamento Dinâmico permite que cada processo lógico decida qual estratégia de Cancelamento, Agressivo ou Preguiçoso, vai utilizar. Mais detalhes podem ser encontrados em (Radhakrishnan et al. 1997, Wilsey 1997).

Outra abordagem utilizada para cancelamento de mensagens é o Cancelamento em Tentativa, utilizado no *Tentative Time Warp*. Nessa abordagem um grupo de mensagens enviadas para um processo lógico pode ser cancelado por uma única antimensagem.

3.5.1.5 Classificação por *Fossil Collection*

Como explicado no capítulo 2, os processos lógicos *Time Warp* necessitam guardar as informações referentes aos valores de suas variáveis de estado para poder desfazer os efeitos de execuções errôneas. Conforme o tempo simulado é avançado pelos processos lógicos, as informações referentes a alguns estados e eventos executados (denominados *fossils*) não são mais necessárias para efetuar *rollback*.

Nesse caso, podem ser descartadas através do procedimento denominado *fossil collection* (figura 3.8).

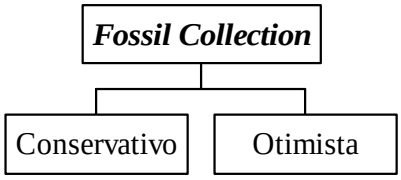


Figura 3.8: Classificação por *Fossil Collection*.

As versões tradicionais do *Time Warp* implementam *fossil collection* através da comparação da marca de tempo da informação que foi salva com o valor do *GVT*. Tudo o que for anterior a esse tempo pode ser descartado e a memória utilizada pode ser liberada, uma vez que os *rollbacks* só irão ocorrer no máximo até o valor do relógio global. Isso implica que o cálculo do *GVT* deve ser efetuado regularmente (ou sempre que um processo lógico necessita espaço em memória) para que os processos lógicos consigam liberar espaço de memória e a simulação possa prosseguir (Young et al. 1996, Young et al. 1999). Essa abordagem pode ser considerada conservativa uma vez que os processos lógicos somente liberam a memória ocupada depois de ter certeza de que os estados não serão mais necessários para garantir a correta execução dos eventos da simulação e isso só acontece quando o valor de *GVT* é calculado. Um processo lógico que utiliza mais espaço em memória do que os demais e precisa liberá-lo pode afetar o desempenho da simulação com sucessivos pedidos para que o cálculo do *GVT* seja efetuado.

A abordagem *OFC* (*Optimistic Fossil Collection*) permite que os processos lógicos efetuem a tarefa de liberar espaço de memória de maneira independente, sem trocar informação com os demais processos. Cada um deles faz uma previsão do valor do *GVT* através de análises estatísticas. Young (Young et al. 1999) sugere examinar as marcas de tempo dos eventos e criar um modelo estatístico para a variação do *LVT*. Com isso pode-se determinar uma estimativa do número X de estados de forma que a probabilidade de que um *rollback* atinja um número de estados maior do que X seja um fator de risco (determinado pelo usuário da simulação distribuída). Esse fator de risco é utilizado para determinar a agressividade da tarefa de efetuar *fossil collection* (Young et al. 1996, Young et al.

1999). Dessa forma, a liberação de espaço em memória é feita de forma otimista, pois cada processo lógico considera que os estados mais antigos não serão mais necessários e, ao contrário da abordagem conservativa, não fica esperando o cálculo do *GVT*.

O comportamento de *rollbacks* de cada processo lógico não pode ser determinado de forma precisa porque é necessário ter informações sobre os estados dos outros processos lógicos, ou seja, não se pode determinar o valor de *GVT* somente com a informação disponível localmente. Devido a esse fator, existe chance de que uma informação jogada fora por um processo lógico venha a ser necessária para corrigir um erro de causa e efeito (falha no mecanismo *OFC*). Isso pode acontecer porque os estados salvos são desprezados com base em uma análise estatística local.

Para evitar que a simulação pare por não encontrar um estado salvo na Lista de Estados (o mecanismo de *rollback* não irá funcionar nesse caso, pois o estado necessário foi desprezado), de tempos em tempos todo o estado da simulação é armazenado (tarefa esta denominada *checkpointing*). Um mecanismo de recuperação permite a reconstrução dos estados desprezados erroneamente através do reinício da simulação a partir de um estado globalmente consistente e com marca de tempo mais próxima da marca de tempo relativa ao erro (que pode ser, no pior caso, o estado inicial da simulação distribuída).

O objetivo desse método otimista de *fossil collection* é reduzir a sobrecarga de mensagens trocadas pelos processos lógicos para o cálculo do *GVT* (Young et al. 1998), especialmente nos casos onde um processo lógico apresenta comportamento diferente em relação aos demais, no que diz respeito às necessidades por espaço em memória. Porém, a parte de recuperação de uma falha no *OFC* exige que os processos lógicos entrem em consenso no momento de se determinar o ponto de *checkpointing* correto para retornar (alguns processos lógicos podem estar mais avançados no tempo simulado do que outros), de modo parecido com o que ocorre nos algoritmos de cálculo de *GVT*.

3.5.2 Classificação Segundo o Mecanismo de Sincronização

Existem muitos protocolos que buscam limitar o otimismo no *Time Warp*, isto é, procuram evitar que os processos lógicos executem eventos de forma desenfreada. Com isso buscam diminuir a ocorrência de erros de causa e efeito e melhorar o desempenho da simulação distribuída.

A taxonomia proposta agrupa os protocolos conforme apresentado na figura 3.3(b). O protocolo Básico consiste no *Time Warp* proposto por Jefferson (Jefferson 1985) sem nenhuma modificação ou otimização quanto ao otimismo. Os protocolos Estendidos consistem em variantes do básico *Time Warp* que procuram reduzir o número de erros de causa e efeito e assim melhorar o desempenho da simulação distribuída.

Os protocolos Estendidos procuram aproveitar as vantagens do *Time Warp* básico como, por exemplo, explorar o paralelismo inerente à aplicação e também eliminar ou ao menos amenizar suas desvantagens (como o excesso de otimismo que pode levar a computações errôneas). A idéia básica consiste em limitar esse excesso de confiança de que a computação de um evento está sendo efetuada corretamente no tempo e assim diminuir a quantidade de erros de causa e efeito e até mesmo a necessidade por espaço livre em memória para armazenar os estados salvos. Assim como o *Time Warp* básico, todos os protocolos que estendem a funcionalidade do *Time Warp* fazem uso das Características de Implementação descritas anteriormente.

Os protocolos Estendidos podem ser divididos em três grandes grupos, que envolvem aqueles que introduzem técnicas para limitar o otimismo na sincronização entre os processos lógicos, aqueles que se preocupam em otimizar o consumo do espaço de memória disponível para que os processos lógicos efetuem salvamento de estados (limitando assim indiretamente o otimismo), e aqueles que se preocupam com o escalonamento de processos lógicos que executam em um mesmo elemento de processamento (o processo lógico com menor chance de sofrer *rollback* pode executar) (figura 3.9). Essas duas últimas visões dos protocolos podem ser utilizadas em conjunto com a limitação do otimismo.

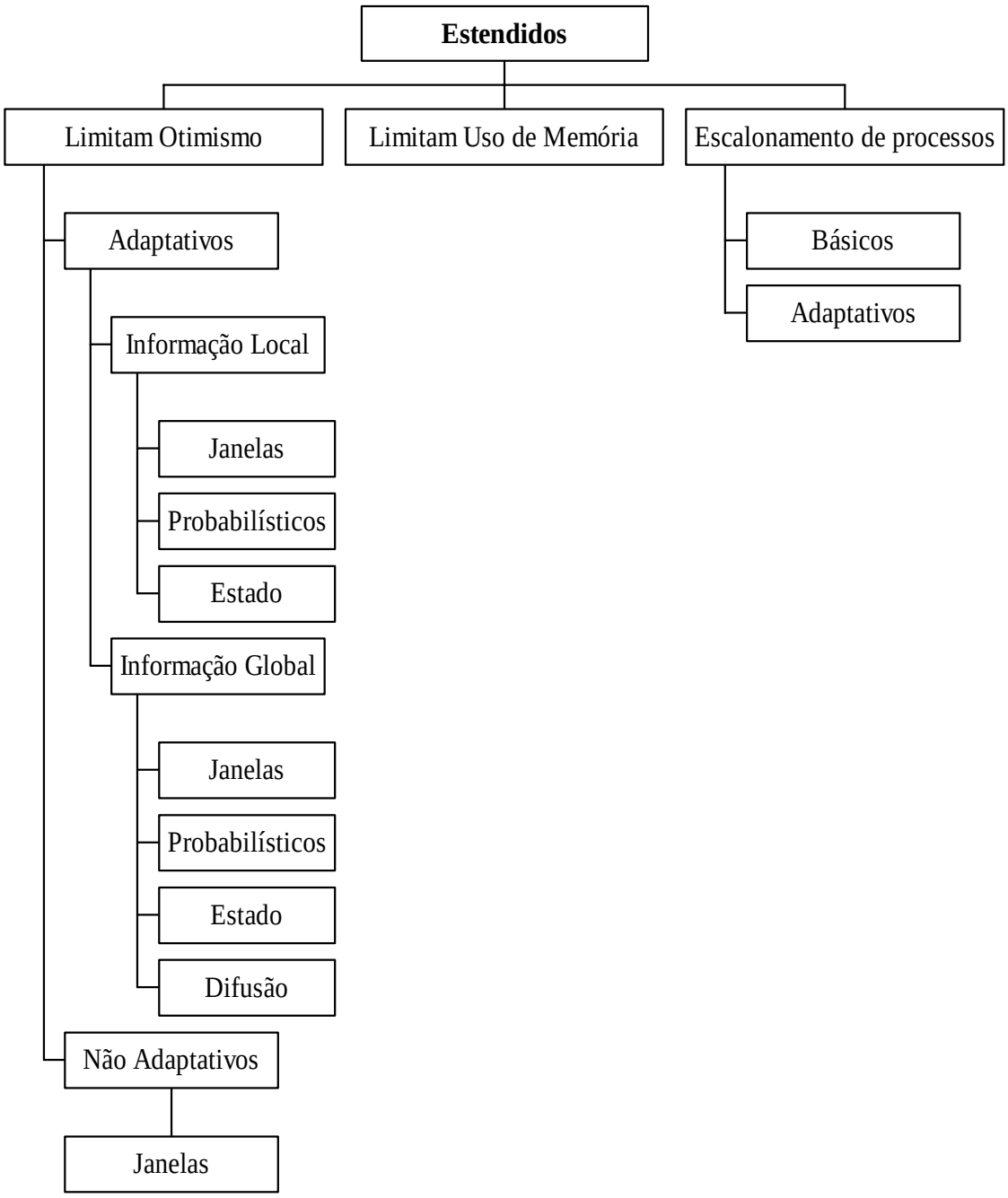


Figura 3.9: Classificação Segundo o Mecanismo de Sincronização (Protocolos Estendidos).

3.5.2.1 Protocolos que Limitam o Otimismo

Esses protocolos podem ser classificados de acordo com a maneira pela qual limitam o otimismo e pela forma com que a técnica utilizada para limitação se comporta no decorrer da simulação distribuída (isto é, se existe adaptação em relação

a mudanças no comportamento da simulação ou não) resultando em protocolos Adaptativos ou Não Adaptativos (Das 1996).

A maioria dos protocolos existentes pode ser classificada como adaptativa, isto é, utilizam a situação corrente da simulação distribuída para decidir quando o otimismo deve ser controlado.

Os vários protocolos Adaptativos diferem no conjunto de informações que são utilizadas para avaliar o comportamento da simulação até o momento, isto é, quais dados e como esses dados são utilizados. Dessa forma, os protocolos podem ser divididos em Locais (a tomada de decisão leva em consideração somente as informações locais ao processo lógico) ou Globais (a decisão em um processo lógico considera informações relativas a outros processos lógicos).

Os protocolos Não Adaptativos limitam o otimismo de forma estática, isto é, não existe alteração durante a execução da simulação distribuída. Alguns mecanismos baseados em janelas se encaixam nesta categoria.

Dependendo da implementação, diferentes protocolos podem ser classificados como Locais ou Globais ou mesmo como sendo Não Adaptativos, como mostrado na figura 3.10 e discutido a seguir.

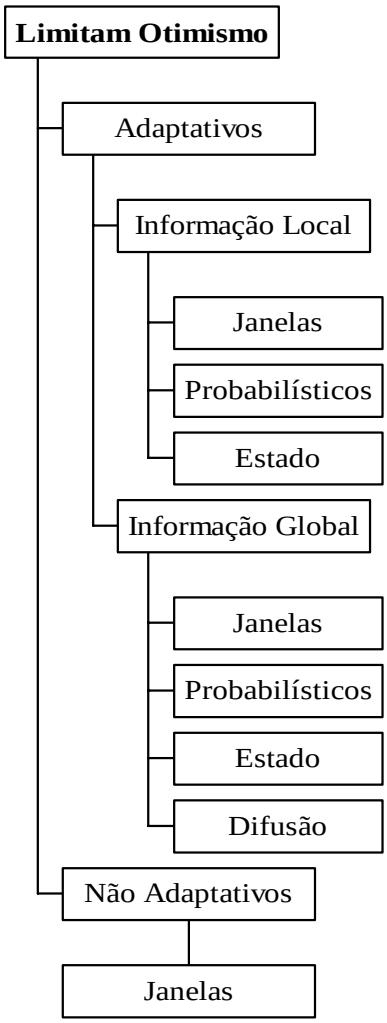


Figura 3.10: Classificação Segundo o Mecanismo de Sincronização (Protocolos que Limitam Otimismo).

Protocolos Baseados em Janelas de Tempo

Os protocolos com base em janelas de tempo permitem que os processos lógicos executem somente aqueles eventos cujas marcas de tempo são inferiores a um determinado valor limite (limite da janela de tempo). Exemplos de implementações não adaptativas incluem *Bounded Time Warp* (Ferscha 1995a; Turner 1992a), *Moving Time Windows* (Ferscha 1995a) e *Window-based Throttling* (Reiher; Jefferson 1989). Exemplos de protocolos onde os limites das janelas se adaptam dinamicamente às condições da simulação incluem *Adaptive Time Warp* (Ferscha 1995a), *Breathing Time Warp* (Steinman 1993), *Adaptive Bounded Time Window* (Das 1996), *UDS* (Rajaei et al. 1993) e *Adaptive Throttle Scheme* (Tay et al. 1997).

Os protocolos baseados em janelas diferem com relação ao método utilizado para determinar o tamanho da janela e se os processos lógicos compartilham a mesma janela (informação global) ou se existe uma janela particular para cada um (informação local). Como exemplo de protocolo com janelas locais tem-se *Breathing Time Warp* e protocolos com janelas globais têm-se *Window-based Throttling* e *Bounded Time Warp.*

Protocolos Probabilísticos

Esses protocolos se baseiam em análises probabilísticas para prever o comportamento futuro dos processos lógicos e assim tomar decisões sobre a sincronização dos mesmos. Como exemplo pode-se citar *MIMDIX*, Protocolo Probabilístico baseado em Custos, Protocolo CO-OP, Protocolo *Probabilistic Distributed Discrete Event Simulation* (Ferscha 1995a) e o Protocolo *PADOC.*

Protocolos Baseados no Estado

Outra classe de protocolos otimistas baseados no *Time Warp* compreende a que se baseia no estado da simulação. Dependendo das condições da simulação os processos lógicos são impedidos de executar eventos e assim não avançam seus *LVTs*. Como exemplo tem-se *Penalty based Throttling*, *Length-based Blocking* e *Composite Elsa,* nos quais as informações de estado são utilizadas localmente por cada processo lógico, e *State Query Time Warp* e os Protocolos Adaptativos *NPSI* (Srinivasan, Reynolds Jr. 1995a), nos quais as decisões baseiam-se no estado global da simulação. É importante ressaltar que o protocolo *Penalty based Throttling* é definido por Das como baseado em Estado local, e por Snirivasan como baseado em Penalidade. Nesta taxonomia, entende-se que a penalidade sofrida por um processo lógico está associada a informações relativas ao seu estado e, portanto, justifica-se a inclusão do protocolo como baseado em estado.

Protocolos Baseados em Difusão

Esses protocolos têm como princípio básico a difusão de informação entre os processos lógicos, com o objetivo de parar computações errôneas. Pode-se considerar que tais protocolos são de abordagem global, porque a informação difundida pode afetar todos os processos da simulação, e adaptativos, porque seu

comportamento pode mudar conforme a simulação apresenta mais ou menos erros de causa e efeito. Como exemplo tem-se os protocolos *Wolf*, *Filter* e *SFilter*.

Os protocolos *Wolf* e *Filter*, classificados por Snirivasan como baseados em conhecimento, são aqui classificados como baseados em difusão. Das os classifica apenas com limitantes do otimismo no *Time Warp*, sem aprofundar a maneira pela qual esta limitação acontece.

3.5.2.2 Protocolos que Controlam o Uso do Espaço de Memória

Essa classificação inclui as técnicas de limitar o uso do espaço de armazenamento disponível, isto é, o progresso dos processos lógicos no tempo de simulação é controlado de acordo com a disponibilidade de espaço em memória. Isso se deve ao fato que os mecanismos de sincronização que têm como base a proposta de Jefferson necessitam salvar os estados dos processos lógicos, para poder desfazer a execução de eventos executados fora de ordem cronológica, e esse espaço pode se esgotar. Esses mecanismos podem ser utilizados em conjunto com os protocolos que limitam otimismo.

Ressalta-se que Das classifica os protocolos de gerenciamento de memória como protocolos adaptativos baseados no estado global da simulação. Na taxonomia proposta neste capítulo, preferiu-se agrupar os protocolos de controle da memória disponível aos processos lógicos em um grupo à parte. Todos eles são adaptativos e tomam decisões com base no estado da simulação (falta de espaço para armazenar novos estados), porém são os únicos que se preocupam com a memória finita da plataforma.

A aplicação desses mecanismos é atrativa devido a vários motivos. Primeiro, porque a utilização de espaço de armazenamento é um ponto crítico nos protocolos otimistas. Segundo, porque eles efetuam um controle indireto sobre o progresso da simulação, evitando *rollbacks*. Exemplos desses mecanismos são *Adaptive Memory Management Protocol* e *Cancelback Protocol* (Ferscha 1995a).

É importante ressaltar que o interesse deste trabalho reside nos protocolos que limitam explicitamente o otimismo no *Time Warp* e, por isso, os mecanismos que

gerenciam o espaço de armazenamento disponível para os processos lógicos não são detalhados.

3.5.2.3 Protocolos que Utilizam Escalonamento de Processos Lógicos

A principal forma de se evitar que os processos lógicos no *Time Warp* executem eventos de forma totalmente otimista é através do controle direto desse otimismo, através dos protocolos discutidos na seção 3.5.2.1. O tratamento indireto desse problema também tem sido estudado na literatura, como é o caso do escalonamento de processos lógicos.

Em simulações distribuídas que envolvem um grande número de processos lógicos, é provável a atribuição de mais de um processo lógico para um mesmo elemento de processamento (Cortellessa; Quaglia 2000, Quaglia, Cortelessa 2000). Assim, pode-se utilizar um algoritmo de escalonamento de processos lógicos de forma que aqueles com menores chances de executarem eventos fora de ordem tenham prioridade.

Esses algoritmos de escalonamento podem ser divididos em Básicos ou Estendidos (figura 3.14). Os básicos caracterizam-se pelo fato de escalonarem sempre o processo lógico com menor valor de *LVT* (*Lowest-Local-Virtual-Time-First*) ou aquele com menor valor de marca de tempo do próximo evento a ser executado (*Lowest-Timestamp-First*).

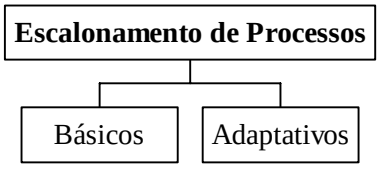


Figura 3.11: Classificação Segundo o Mecanismo de Sincronização (Escalonamento).

Os algoritmos estendidos procuram avaliar qual processo lógico tem menor chance de sofrer *rollback* ao executar o próximo evento. Este processo lógico é escolhido para executar. Exemplos incluem *Adaptive Control based Scheduling, Service Oriented Scheduling, Probabilistic Scheduling, State based Scheduling, Grain Sensitive Scheduling* e *Aggressiveness/Risk Effects based Scheduling*. Mais

detalhes referentes a esses algoritmos podem ser encontrados em (Quaglia; Cortellessa 2000).

3.5.3 Classificação Segundo o Modo de Limitar o Otimismo

Vários protocolos de sincronização descritos na literatura modificam o *Time Warp* original de forma que os processos lógicos limitem o excesso de otimismo. Dessa forma, os processos lógicos podem moldar o seu comportamento em virtude das características da simulação, como por exemplo o aumento do número de *rollbacks*, ou então em virtude de uma característica que tenta prevenir a ocorrência de *rollbacks* sem analisar esse número (Pham; Dida 1996).

Pode-se classificar os protocolos limitantes do *Time Warp* em Preventivos e Corretivos (figura 3.3(c)). Os primeiros se propõem a prevenir a ocorrência de *rollbacks*, como por exemplo *Length-based Blocking* e os protocolos baseados em Janelas Não Adaptativos. Os protocolos Corretivos, como o próprio nome indica, corrigem o comportamento do processo lógico em função do aumento do número de erros de causa e efeito (é importante não confundir com a correção do erro de causa e efeito, associada a tarefa do *rollback*). Como exemplo têm-se *Penalty-based Throttling* e os protocolos Adaptativos.

3.6 Considerações Finais

Este capítulo apresentou os principais protocolos que inserem variações no *Time Warp* básico proposto por Jefferson (Jefferson 1985), algumas classificações disponíveis na literatura e a proposta de uma nova taxonomia. O grande número de variantes torna difícil o trabalho de escolha de um determinado protocolo quando se deseja utilizar a simulação distribuída para resolver uma aplicação.

Para facilitar essa escolha, pode-se efetuar uma comparação de desempenho entre os mecanismos e utilizar aquele com melhor resultado. Porém, para realizar essa tarefa, é necessário primeiro que se tenha pleno conhecimento dos protocolos que se quer avaliar.

O primeiro passo para efetuar uma avaliação entre protocolos otimistas consistiu em agrupar tais protocolos de acordo com suas características, tarefa

dificultada pelo fato de que alguns protocolos apresentam características das abordagens conservativa e otimista. Ao agrupar os protocolos semelhantes, pôde-se visualizar as diferentes técnicas utilizadas para inserir limitações no otimismo do *Time Warp*.

Existem duas classificações na literatura que buscam agrupar os protocolos de sincronização (apresentadas na seção 3.3). Porém, essas classificações não abrangem protocolos mais recentes e não levam em consideração as características da implementação, que podem influenciar em um estudo de desempenho entre protocolos de sincronização. Além disso, uma delas classifica protocolos que inserem otimismo em protocolos conservativos e ambas tratam os protocolos de gerenciamento de memória juntamente com outros protocolos, apesar das características únicas que apresentam.

Agrupadas e avaliadas as formas de se limitar o otimismo, foi proposta uma taxonomia hierárquica (porque apresenta uma hierarquia entre os protocolos de sincronização) para permitir que a tarefa de avaliação de desempenho seja efetuada. As classificações apresentadas em trabalhos anteriores (Das 1996, Srinivasan; Reynolds 1995a, Srinivasan; Reynolds 1995b) forneceram as idéias básicas e motivaram a nova proposta, que é mais atual e concisa. Os principais protocolos de sincronização descritos na literatura e que têm como objetivo principal limitar o excesso de otimismo no *Time Warp* foram divididos em três grandes grupos, que se preocupam com as Características de Implementação, Mecanismo de Sincronização e o Modo de Limitar o Otimismo. Cada um desses grupos apresenta uma hierarquia de subgrupos.

A taxonomia proposta neste capítulo constitui uma das contribuições relevantes desta tese e sua versão inicial foi apresentada e discutida na *13th European Simulation Multiconference* (Spolon et al. 1999). Essa taxonomia é uma proposta nova e mais completa do que as anteriormente publicadas, no sentido de se preocupar com variantes do *Time Warp* e também com as características de implementação e modo de limitar o otimismo, que não são abordados nas outras classificações.

O próximo capítulo apresenta uma discussão sobre a aplicabilidade das técnicas de avaliação de desempenho de sistemas computacionais à análise de protocolos de sincronização e apresenta um modelo representativo do comportamento de um processo lógico *Time Warp* básico.

Capítulo 4

Modelo de Um Processo Lógico *Time Warp*

Este capítulo apresenta a descrição do comportamento de um processo lógico que utiliza o protocolo *Time Warp* como protocolo de sincronização da simulação distribuída. A partir dessa descrição foi desenvolvido e validado um novo modelo básico para um processo lógico, especificado em Redes de Petri. Esse modelo representa uma contribuição relevante para a área de simulação distribuída, devido à forma abrangente e completa com que o assunto é abordado.

4.1 Considerações Iniciais

No capítulo 3 foi apresentado um grande número de protocolos que modificam a funcionalidade do *Time Warp* e proposta uma nova taxonomia para organizar essa variedade de protocolos. De posse dessa nova taxonomia, o usuário da simulação distribuída pode compreender quais estratégias os protocolos otimistas utilizam para modificar o excesso de otimismo dos processos lógicos *Time Warp*. Porém, simplesmente analisar a taxonomia e olhar as vantagens e desvantagens de cada subclasse de protocolos não é suficiente para resolver o problema de como escolher um desses protocolos. Dessa maneira, é interessante que as pessoas disponham de alguma forma de avaliação que permita comparar os protocolos existentes e auxilie no processo de tomada de decisão sobre qual protocolo utilizar.

Diferentes trabalhos foram desenvolvidos para efetuar a avaliação de desempenho de simulação distribuída (Spolon et al. 1999). A maior parte dos

trabalhos utiliza casos específicos de teste, sendo que um aspecto negativo dessas abordagens está relacionado ao fato de que é preciso ter um protocolo implementado para que se possa experimentá-lo. Outros trabalhos propõem *benchmarks* e há também estudos analíticos que são complexos, o que pode tornar a análise uma tarefa árdua e muitas vezes inacessível para um usuário.

A representação do *Time Warp* em um modelo e a solução desse modelo, através de simulação, torna-se atrativa quando se deseja inserir modificações e obter resultados. O *Time Warp* é um protocolo complexo e a sua representação em um modelo é uma tarefa que exige profundo conhecimento dos seus sub-algoritmos (como discutido no capítulo 3). A definição da taxonomia apresentada no capítulo 3 auxiliou na identificação desses sub-algoritmos, criando domínios de *plugins*. Cada domínio é responsável por uma atividade e cada elemento do domínio é responsável pela implementação de um sub-algoritmo (*plugin*).

Neste capítulo apresenta-se o estudo pormenorizado do comportamento de um processo lógico *Time Warp,* composto da identificação de suas atividades principais e a definição de um modelo, representado através de Redes de Petri. As Redes de Petri têm como característica o fato de permitirem a representação de atividades concorrentes (Jensen 1998) e um alto grau de detalhamento nas características do sistema em estudo e, por isso, foram utilizadas.

Este capítulo consiste em uma contribuição importante desta tese devido à maneira abrangente e completa com que é apresentado um processo lógico *Time Warp* básico. A descrição informal do *Time Warp* encontrada na literatura não apresenta um nível de esclarecimento suficiente para uma implementação ou o desenvolvimento de um modelo, como o produzido nesta tese e detalhado neste capítulo.

A seção 4.2 discute a aplicação das técnicas de avaliação de desempenho em sistemas computacionais para a análise de protocolos de sincronização em simulação distribuída e justifica o uso da modelagem como técnica de avaliação nesta tese. A seção 4.3 apresenta a metodologia aplicada no desenvolvimento de um estudo de desempenho através de técnicas de modelagem. A seção 4.4 mostra a aplicação dessa metodologia no desenvolvimento do modelo de um processo lógico *Time Warp*. Na seção 4.5 são apresentados alguns *plugins* e é discutido como eles podem

ser adicionados ao modelo básico de um processo lógico. A seção 4.6 apresenta as considerações finais do capítulo.

4.2 Avaliação de Desempenho de Protocolos de Sincronização para Simulação Distribuída

As técnicas de avaliação de desempenho de sistemas computacionais têm sido adotadas no estudo da simulação distribuída. A avaliação de desempenho de um sistema computacional exige algum tipo de observação para que se possam obter as informações necessárias ao estudo. Essas informações podem ser obtidas diretamente no sistema (técnicas de aferição) ou através de experimentação com um modelo que representa o sistema em estudo (técnicas de modelagem), como sintetizado na tabela 4.1 (Francês et al 1997).

Tabela 4.1: Técnicas para Avaliação de Desempenho de Sistemas Computacionais.

Técnicas de Aferição	Técnicas de Modelagem
Coleta de Dados	Solução Analítica
Benchmarks	Solução através de Simulação
Protótipos	

A coleta de dados pode ser efetuada através do uso de monitores de hardware ou de software. Um *benchmark* consiste em um programa (ou conjunto de programas) escrito em uma linguagem de alto nível e representando uma classe de aplicações. Já um protótipo é uma versão simplificada do sistema real.

As técnicas de aferição através de coleta de dados e *benchmarks* são adequadas para sistemas existentes, sendo que os *benchmarks* são largamente utilizados para análises comparativas. Os protótipos, entretanto, podem representar uma solução atrativa para avaliação de sistemas em desenvolvimento ou quando a observação do sistema real não é viável, porém podem exigir grandes investimentos.

As técnicas de modelagem podem ser adotadas para sistemas reais ou inexistentes. Um modelo consiste em uma abstração das características do sistema, que têm importância no estudo que vai ser desenvolvido e é utilizado como um veículo para se realizar experimentos e responder a perguntas sobre o sistema. A construção de um modelo de um sistema pode ser efetuada textualmente através de

técnicas como, por exemplo, Estelle - *Extended State Transition Language* (Budkowski; Dembinski 1987) ou através de alguma técnica gráfica, como por exemplo as Redes de Fila (MacDougall 1987), Redes de Petri (Petri 1966), *Statecharts* (Harel et al. 1987) e Máquinas de Estado Finito (Gill 1962). A dificuldade de se ter uma solução analítica aumenta com a complexidade do modelo. Além disso, alterações no modelo não são tão simples de serem refletidas na solução. O uso de simulação como solução para o modelo pode ser mais simples que a solução analítica, além de permitir de modo mais fácil a introdução de alterações do modelo.

A avaliação de desempenho de protocolos de sincronização em simulação distribuída também pode ser efetuada utilizando-se as mesmas técnicas utilizadas para sistemas computacionais. Isto é, é possível utilizar as técnicas de aferição e modelagem. A adoção de técnicas de aferição apresenta as mesmas vantagens e desvantagens discutidas para os sistemas computacionais, uma vez que exige que o protocolo (ou uma versão simplificada) esteja implementado para poder ser experimentado. As técnicas de modelagem, por outro lado, podem permitir, por exemplo, a identificação de fatores que degradam o desempenho de um novo protocolo de sincronização (em desenvolvimento), sem que haja a necessidade de implementá-lo. Outro aspecto importante é a possibilidade de prever o comportamento de um novo protocolo antes de desenvolver o código para resolver uma aplicação particular. Essas características das técnicas de modelagem fornecem a oportunidade de escolher o protocolo mais adequado para uma aplicação em particular.

As técnicas mais comumente utilizadas envolvem a aferição através dos *benchmarks* (como por exemplo Pucks (Bellenot 1993), *PHOLD* (Fujimoto 1990b) e *Incremental Benchmark Suite* (Rönngren et al. 1996b)) e a construção de protótipos, como por exemplo *Spectrum* (Reynolds Jr. 1989), *N_MAP* (Ferscha; Johnson 1996) e *WSL* (Balakrishnan et al. 1997). Porém, na maior parte dos casos apresentados na literatura os autores desenvolvem seus próprios casos de teste (Jha; Bagrodia 1996, Chiola; Ferscha 1995, Fabbri; Donatiello 1997, Porras et al. 1998). Existem também algumas soluções analíticas de modelos do *Time Warp* (Gupta et al. 1991; Nicol 1991; Srinivasan; Reynolds Jr. 1995b; Tay et al. 1998). Ulson (Ulson 1999a) faz experimentações sobre o protocolo de simulação distribuída *CMB*, implementado em

diferentes plataformas; Morselli (Morselli 2000) utiliza a simulação para representar o comportamento do mesmo protocolo.

Assim, as técnicas de modelagem são adequadas ao estudo de um sistema inexistente, ou em situações onde a aferição (por meio de *benchmarks* ou coleta de dados) pode interferir nos resultados obtidos. Dessa forma, são as técnicas que melhor se adaptam aos objetivos deste trabalho. Além disso, ressalta-se a inviabilidade de se implementar toda a gama de protocolos otimistas apresentados no capítulo 3; também é inviável ter todos os protocolos em versões executáveis em diferentes ambientes computacionais.

A seção 4.3 descreve como utilizar as técnicas de modelagem para solucionar um problema de avaliação de desempenho.

4.3 Metodologia Utilizada Para o Desenvolvimento de Um Estudo de Desempenho Através de Técnicas de Modelagem

O estudo de desempenho através das técnicas de modelagem é composto de vários passos que englobam o desenvolvimento do modelo, os testes para garantir que está correto e a obtenção dos resultados através da experimentação do modelo.

O primeiro passo em um estudo de avaliação de desempenho, independente da técnica utilizada para resolver o modelo, consiste em identificar o problema que gerou a necessidade de avaliação de desempenho. Feito isso deve-se analisar o sistema que vai ser avaliado no estudo e estabelecer os objetivos desejados. A seção 4.4.1 descreve esses passos, aplicados ao estudo do *Time Warp* básico.

De posse dos objetivos da análise de desempenho o sistema é cuidadosamente estudado para que se possam abstrair as características fundamentais para a construção de um modelo representativo. Uma tarefa relativamente difícil nesta fase consiste na tomada de decisão sobre quais elementos do sistema devem ser incluídos no modelo. O nível de detalhamento deve basear-se no propósito para o qual o mesmo está sendo construído. A seção 4.4.2 detalha esses passos.

O passo de formulação do modelo gera os requisitos para os dados de entrada que servirão como parâmetros do mesmo. Quando a modelagem é efetuada sobre um sistema existente, os parâmetros do modelo podem ser medidos, caso contrário devem ser estimados (seção 4.4.3).

Deve-se então escolher a técnica para solução do modelo, conforme apresentado na seção anterior. A escolha da simulação envolve o desenvolvimento de uma representação computacional do modelo (seção 4.4.4), o qual deve ser conferido para garantir que está livre de erros de programação e de lógica. A verificação compara a representação computacional com o modelo visando garantir que o modelo foi fielmente representado. Não existem regras específicas para efetuar essa tarefa, mas sim algumas abordagens que podem ser seguidas, como por exemplo inspeção, comparando o programa de simulação e o modelo (Higginbotton 1998). O último passo antes do início da experimentação do modelo para a obtenção dos resultados envolve a validação, que é utilizada para mostrar que o modelo representa o sistema em estudo, ou seja, que reproduz seu comportamento (seção 4.4.5). Quando o sistema em estudo existe e pode ser utilizado para medições, a validação pode se basear na comparação dos resultados do modelo com aqueles obtidos nas medições efetuadas no sistema real. Se o sistema não existe é preciso utilizar alguma forma de validação do modelo conceitual. A literatura oferece algumas alternativas (SA84, MA87 *apud* Santana 1989a), entre elas *face validity*, valores fixos, limites assintóticos e validação interna.

Verificado e validado o modelo, pode-se experimentá-lo para obter os resultados desejados e selecionam-se as medidas (as métricas) que serão utilizadas para avaliar o desempenho (capítulo 5). Deve-se ter cuidado ao obter os resultados da simulação utilizando-se técnicas de análise de saída que auxiliam na obtenção de uma estimativa precisa de uma medida de desempenho.

4.4 Desenvolvimento do Estudo de Desempenho

As seções seguintes descrevem o desenvolvimento do estudo de desempenho através de um modelo representativo do comportamento de um processo lógico *Time Warp*, segundo os passos do ciclo de vida de uma simulação. O primeiro passo do ciclo de vida foi iniciado no capítulo 2, com uma breve descrição do sistema em estudo, o protocolo de sincronização *Time Warp*.

4.4.1 Identificação do Problema e Objetivos do Estudo

Neste trabalho o sistema em estudo consiste no protocolo de sincronização para simulação distribuída otimista *Time Warp* e variantes, compostos de vários processos lógicos que trocam mensagens e antimensagens com o objetivo de manter a relação de causa e efeito entre os eventos da simulação executados. Como esse comportamento é similar em todos os processos lógicos, restringe-se neste capítulo o estudo ao sub-sistema constituído por um único processo lógico.

O objetivo do estudo de desempenho é construir um método que permita a avaliação dos protocolos *Time Warp* e variantes, oferecendo ao usuário da simulação distribuída otimista uma maneira objetiva de escolher o protocolo que melhor se adapta à solução da sua aplicação.

4.4.2 Identificação das Características Importantes para a Construção do Modelo do Comportamento do *Time Warp*

Definida a meta do estudo de desempenho, o passo seguinte corresponde à identificação das características do sistema (no caso, um processo lógico do protocolo *Time Warp*), que apresentam impacto no desempenho do mesmo e que, portanto, devem ser incluídas em um modelo representativo. Essa tarefa exige familiaridade com o protocolo e um conhecimento mais profundo do que a descrição elementar encontrada na literatura.

Dessa forma, foi necessário identificar e definir todas as tarefas executadas por um processo lógico. Uma das dificuldades em se desenvolver um modelo – uma abstração – do comportamento básico de um processo lógico *Time Warp* reside no fato de que não existem descrições detalhadas do mesmo na literatura. A proposta de tempo virtual de Jefferson (Jefferson 1985) explica a noção de processos lógicos e a interação entre eles, porém não define como as atividades devem ser implementadas. Como mostrado no capítulo 3, isso abre um grande leque de possibilidades que torna muito difícil a comparação dos protocolos de sincronização, uma vez que cada um implementa as diretivas do tempo virtual de forma diferente, buscando otimizações em um ou outro aspecto.

O protocolo *Time Warp* gerencia a comunicação e a sincronização entre os processos da simulação distribuída. Cada processo lógico é responsável pela execução de uma parte dos eventos, que pode gerar novos eventos para execução local ou remota. As antimensagens e as mensagens com marca de tempo menor do que o valor do *LVT* provocam *rollbacks*. A figura 4.1 (Spolon et al. 2000a) apresenta uma abstração de um processo lógico em um diagrama de blocos que divide as tarefas entre recepção e execução.

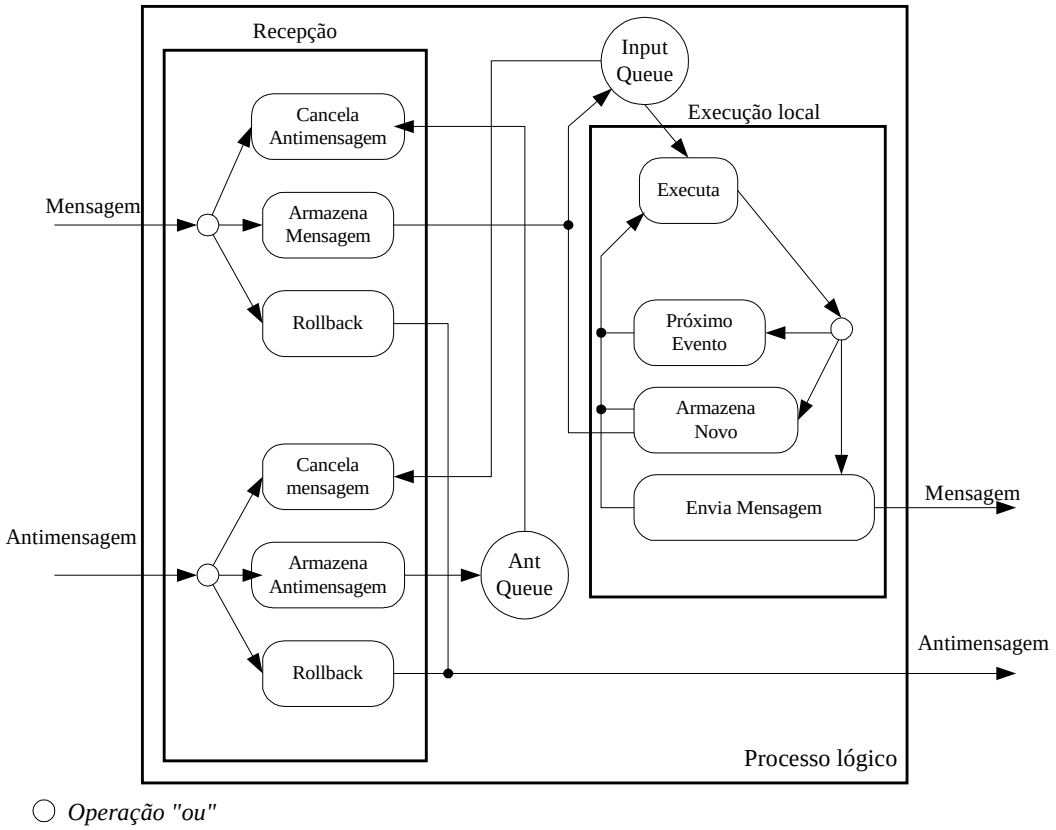


Figura 4.1: Abstração de um Processo Lógico *Time Warp*.

Um processo lógico *Time Warp* lida com a recepção e a execução de mensagens e antimensagens, as quais podem provocar o envio de novas mensagens e antimensagens. A recepção de mensagens e antimensagens enviadas por processos lógicos remotos é representada pelo bloco *Recepção.*

As seguintes ações são tomadas quando do recebimento de uma mensagem:

- O processo lógico deve verificar se recebeu anteriormente alguma antimensagem correspondente à mensagem que chegou. Em caso

afirmativo a antimensagem é removida da *Input Queue* e ambas são descartadas (bloco *Cancela Antimensagem*);

- Caso o processo lógico não tenha recebido nenhuma antimensagem cuja marca de tempo corresponda à mensagem, a marca de tempo da mesma é comparada com o *LVT*:
    - Se a marca de tempo é maior ou igual ao *LVT* então a mensagem é armazenada na *Input Queue* e será executada oportunamente (bloco *Armazena Mensagem*);
    - Se a marca de tempo da mensagem recebida é menor do que o valor do *LVT*, ocorre um erro de causa e efeito e o processo lógico sofre um *rollback* para voltar a um estado consistente. Nesse caso podem ser enviadas antimensagens para corrigir computações errôneas enviadas a outros processos lógicos (bloco *Rollback*).

Quando o processo lógico recebe uma antimensagem pode acontecer uma das seguintes situações:

- Se a antimensagem chegou antes da mensagem correspondente, ela é armazenada na Lista de Antimensagens (bloco *Armazena Antimensagem*);
- Se a mensagem correspondente à antimensagem já foi recebida pelo processo lógico mas ainda não foi executada, ambas são descartadas (bloco *Cancela Mensagem*);
- Se a mensagem correspondente à antimensagem foi recebida e já foi executada pelo processo lógico, uma operação de *rollback* deve ser executada para retornar a computação a um estado seguro. Isso pode envolver o envio de novas antimensagens (bloco *Rollback*).

O bloco de execução local mostras as três ações relacionadas com a criação de novos eventos desenvolvidas para a execução de um novo evento no *Time Warp*. O processo lógico remove a mensagem que contém a menor marca de tempo da *Input Queue* e executa o evento correspondente. As seguintes ações podem ser efetuadas:

- Um novo evento pode ser gerado para executar em um processo lógico remoto, o que implica no envio de uma mensagem (bloco *Envia Mensagem*);
- Um novo evento pode ser gerado para executar localmente e é inserido na Input Queue para posterior execução (bloco *Armazena Novo*);
- É também possível que o evento executado não produza outro evento (bloco *Próximo Evento*).

Em todos os casos, após a execução do evento e a geração de um novo (se necessário), o processo lógico busca a próxima mensagem da *Input Queue* para executar o evento correspondente.

Analisando o diagrama apresentado na figura 4.1, pode-se observar que os pontos de interesse em um processo lógico *Time Warp* residem na manipulação de mensagens e antimensagens e na execução dos eventos correspondentes, porém a figura não mostra o sequenciamento dessas atividades e não apresenta a manipulação das estruturas de dados que o protocolo necessita. O processo fica continuamente olhando o *Buffer* de Entrada, que guarda as mensagens/antimensagens que chegam pela rede de comunicação, e executando os eventos da *Input Queue*. A figura 4.2 apresenta um algoritmo que resume essas atividades.

```
Enquanto Não Fim da Simulação
   Executa evento da Input Queue
   Gera novos eventos se necessário (locais ou remotos)
   para o Buffer de Saída
   Salva estado
   Recebe mensagens e ou antimensagens que estão no
   Buffer de Entrada
   Rollback se necessário – envia antimensagens através
   de uma das formas de Cancelamento
Fim Enquanto
```

Figura 4.2: Uma Visão Macroscópica das Atividades de um Processo Lógico *Time Warp*.

Tendo como base o diagrama da figuras 4.1 e o algoritmo da figura 4.2, é possível elaborar uma seqüência de passos que devem ser seguidos para a execução de um evento pelo processo lógico. Essa seqüência está representada na figura 4.3. O processo lógico atualiza o *LVT* com a marca de tempo do próximo evento a ser executado (o de menor marca de tempo) e o executa. Como explicado anteriormente, essa execução pode ou não levar à geração de novo evento (isso depende da aplicação que está sendo resolvida).

```
Retira próximo evento da Input Queue
Se Tempo de Salvamento
      Salva estado na State Queue

Caso evento seja
      Não enviar mensagem
            Atualiza LVT
            Insere evento na Input Queue Executados

      Enviar mensagem (remota ou local)
            Atualiza LVT
            Calcula nova marca de tempo
            Decide qual o destino da mensagem
            Insere evento na Input Queue Executados
            Insere antimensagem na Output Queue
            Envia Mensagem (para processo lógico
            remoto) ou local (Buffer de Saída)
```

Figura 4.3: Execução de Evento por Um Processo Lógico.

Caso a execução do evento não implique na geração de um novo evento, o processo lógico salva o estado e busca o próximo evento a ser executado. Se um novo evento é gerado, o destino pode ser o próprio processo lógico ou um processo lógico remoto. A marca de tempo da mensagem que encapsula o evento é calculada e o estado do processo lógico é salvo na *State Queue*. Pode-se utilizar uma das abordagens mostradas no capítulo 3 (*Copy*, *Sparse* ou *Incremental*) para efetuar esta última tarefa. O conteúdo do estado a ser salvo depende da aplicação que está sendo resolvida pela simulação distribuída e do que se pretende com o estudo (isto é, as estatísticas desejadas). Basicamente, o estado envolve os valores das estatísticas, do *LVT*, apontadores das estruturas de fila e variáveis auxiliares da simulação, como por exemplo seqüências e sementes utilizadas nos algoritmos de geração de números pseudo-aleatórios.

Caso o evento executado implique na geração de um novo evento, o processo lógico determina o destino (pode ser ele próprio ou um processo lógico remoto). Uma cópia da mensagem utilizada para encapsular o evento é inserida na *Output Queue*. Essa cópia é necessária para garantir o correto funcionamento do mecanismo de *rollback*. A mensagem é inserida em um *Buffer* de Saída, para ser enviada através da rede de comunicação, ou é inserida no *Buffer* de Entrada do processo lógico (o tratamento para mensagens que chegam da rede de comunicação ou que representam eventos gerados localmente é o mesmo, ambas devem passar pela análise da parte de recepção antes do evento ser inserido na *Input Queue*).

Após a execução do evento e das tarefas descritas anteriormente, o processo lógico o insere na *Input Queue Executados*. É necessário manter uma cópia de todos os eventos executados para o caso de ocorrência de *rollback*.

A figura 4.1 mostra que um processo lógico recebe mensagens e antimensagens as quais podem alterar o fluxo de execução de eventos (podem provocar erros de causa e efeito). Essas mensagens e antimensagens são armazenadas em um *Buffer* de Entrada e analisadas quando o processo lógico não está executando evento. Essas atividades podem ser detalhadas como mostrado na figura 4.4.

```
Retira do Buffer de Entrada
Se é mensagem
      Se a antimensagem está na Ant Queue
            Descartar ambas
      Senão
            Se marca de tempo >= LVT
                  Insere evento na Input Queue
            Senão
                  Rollback
Se é antimensagem
      Se mensagem correspondente ainda não chegou
            Insere antimensagem na Ant Queue
      Senão
            Se mensagem correspondente está na Input
            Queue
                  Se mensagem não foi executada
                        Descartar ambas
                  Senão
                        Rollback
```

Figura 4.4: Recepção de Mensagem e/ou Antimensagem.

Quando não está executando um evento, o processo lógico retira o primeiro elemento do *Buffer* de Entrada e o analisa. Dessa forma, quando se fala em recebimento de mensagens e antimensagens, trata-se da retirada das mesmas do *Buffer* de Entrada. O recebimento de uma mensagem faz com que o processo lógico verifique se a antimensagem correspondente já chegou, ou seja, se está armazenada na *Ant Queue*. Uma busca na estrutura de dados deve ser efetuada e, se a antimensagem for encontrada, é retirada da *Ant Queue*. Ambas são descartadas, sem prejuízo para a execução da simulação. Somente se a operação de busca na *Ant Queue* não for bem sucedida é que o processo lógico compara a marca de tempo da mensagem com o *LVT* e insere o evento correspondente na *Input Queue* ou ativa a operação de *rollback.*

Nem toda antimensagem retirada do *Buffer* de Entrada provoca *rollback*. Isso somente acontece quando a mensagem para a qual a antimensagem foi enviada já foi recebida e executada no processo lógico destino. A busca na *Input Queue* detecta quando o evento ainda não foi executado e ambos, mensagem e antimensagem, são descartados.

Exceto pelas mensagens enviadas e pelo estado que deve ser salvo, a execução de eventos em um processo lógico *Time Warp* consiste no mesmo mecanismo utilizado em uma simulação seqüencial, onde os eventos estão armazenados de forma ordenada em uma fila. Quem toma os cuidados necessários para que a *Input Queue* esteja sempre ordenada é a parte de recebimento de mensagens. Quando ela percebe que entregou um evento fora de ordem para execução (através do recebimento de uma mensagem “atrasada” ou de uma antimensagem), corrige esse erro através do *rollback* (figura 4.5).

Como explicado anteriormente, um processo lógico pode ativar a operação de *rollback* por dois motivos. O mais simples acontece quando uma mensagem com marca de tempo menor do que o *LVT* é recebida (*rollback* primário). O segundo motivo é o recebimento de uma antimensagem, enviada por um processo lógico remoto, devido a ocorrência de um *rollback* (*rollback* secundário).

```
Se rollback causado por mensagem
      Inserir evento na Input Queue
      LVT = marca de tempo do evento
Enquanto não terminou de verificar a Output Queue
      Se TimestampMsgGerouOutput > Timestamp então
            Enviar a antimensagem correspondente
Fim Enquanto
//Procurar todas as msgs executadas erroneamente
Enquanto não terminou de verificar a Input Queue Executados
      Se TimestampInputQueueExecutados > Timestamp
            Inserir evento para reexecução
Fim Enquanto
Restaurar LVTAnterior (do ultimo evento executado)
Se rollback causado por antimensagem
      Restaurar LVT (do último evento executado)
```

Figura 4.5: *Rollback*.

O tratamento que o processo lógico dedica às duas situações é sutilmente diferente e importante porque essa diferença é crucial para o correto funcionamento do protocolo de sincronização. No caso do *rollback* primário, o *LVT* deve ser atualizado com o valor da marca de tempo da mensagem que chegou atrasada, porque o evento correspondente vai ser o próximo a ser executado e é inserido na *Input Queue*. No *rollback* secundário, a antimensagem chega para desfazer uma computação errada e o *LVT* vai ser atualizado com o valor da marca de tempo do último evento executado antes do evento correspondente à mensagem que deve ser neutralizada pela antimensagem.

Em ambas situações a *Output Queue* deve ser pesquisada e todas as mensagens enviadas com marca de tempo maior do que a marca de tempo da mensagem ou da antimensagem que provocou *rollback* devem ser reenviadas em forma de antimensagem.

O último passo do *rollback* envolve a pesquisa na *Input Queue Executados* visando encontrar todos os eventos que foram executados de forma errada para reexecutá-los. Esses eventos são inseridos na *Input Queue* como não executados e dessa forma o protocolo consegue manter a simulação em um estado consistente.

4.4.3 Definição dos Parâmetros do Modelo

Para ser utilizado e produzir os resultados de desempenho esperados, o modelo de simulação deve receber os parâmetros adequados. Como o sistema em estudo consiste em um processo lógico do protocolo de sincronização *Time Warp*, são necessários os parâmetros relativos ao protocolo de sincronização propriamente dito. Esses parâmetros dizem respeito aos algoritmos utilizados para, por exemplo, executar a tarefa de salvamento de estados e a tarefa de cancelamento de mensagens (como apresentado na taxonomia do capítulo 3).

4.4.4 Desenvolvimento do Modelo de Um Processo Lógico *Time Warp* Utilizando Redes de Petri

Conforme apresentado na seção 4.2, há diferentes técnicas para representar um modelo, como por exemplo Redes de Fila, Redes de Petri, *Statechart, Estelle*. As Redes de Fila são mais adequadas em situações onde existam os conceitos de clientes sendo atendidos por um prestador de serviços, mas não permitem a especificação de sistemas concorrentes, como é o caso do protocolo de sincronização *Time Warp*, não sendo assim adequadas.

Tanto as Redes de Petri quanto os *Statecharts* permitem a representação de sistemas concorrentes, porém a representação de filas não é explícita. Nas Redes de Petri existe a permissão para que os lugares sejam genéricos (podem representar tanto recursos de um sistema quanto situações abstratas, como por exemplo o estado de um recurso), os *tokens* podem ser individualizados através do uso de cores e as extensões hierárquicas permitem uma maior compactação do modelo, mesmo que em detrimento da clareza do mesmo (Francês et al. 2000). Além disso, as Redes de Petri possuem tanto representações para transições imediatas (aquelas que são disparadas com tempo zero) quanto para transições temporizadas.

Os *Statecharts*, por sua vez, não individualizam clientes, como as redes de Petri fazem com os *tokens*, e as transições (que são representadas por arcos) são sempre tratadas como imediatas (Francês et al. 2000).

Dessa forma, a representação do modelo do protocolo *Time Warp* seria inviável através das Redes de Filas e ficaria prejudicada com o uso de *Statecharts*,

porque, conforme será explicado na seção 4.5.2 existem atividades que consomem tempo (por exemplo, a execução de um evento) e sua representação é importante para a avaliação do protocolo. Além disso, a distinção entre os *tokens* é importante para a representação das mensagens e antimensagens. As Redes de Petri e Estelle poderiam ser utilizadas para a especificação do modelo, porém, optou-se por utilizar as Redes de Petri na representação do modelo do comportamento de processos lógicos *Time Warp* e variantes pela característica de distinção de *tokens* e hierarquia que as Redes de Petri oferecem.

Existem várias ferramentas que implementam os recursos oferecidos pelas Redes de Petri, porém uma das mais completas, por permitir a representação e a simulação de Redes Coloridas, Hierárquicas e Estocásticas Generalizadas, é o ambiente *ALPHA/Sim*. O *ALPHA/Sim* é uma ambiente de simulação de propósito geral e de domínio privado, que faz uso de propriedades das Redes de Petri Estocásticas Generalizadas, das Redes de Petri Coloridas e das Redes de Petri Hierárquicas, como técnica para efetuar a modelagem. Através de um editor gráfico o *ALPHA/Sim* possibilita ao usuário desenhar e fornecer todos os parâmetros necessários ao modelo de simulação (Alphatech 1995, Moore; Hammer 1999, Moore; Chiang 2000). Maiores detalhes são apresentados no apêndice A.

4.4.4.1 Modelo em Redes de Petri

Os algoritmos que definem as atividades executadas por um processo lógico *Time Warp* apresentados e a taxonomia proposta no capítulo 3 forneceram o embasamento necessário para a construção do modelo do comportamento de um processo lógico *Time Warp*.

A especificação completa do modelo na ferramenta *ALPHA/Sim*, que envolve a lógica da Rede de Petri, os tipos de *tokens* (as cores), o tratamento dos atributos nas transições, e as decisões utilizadas nos lugares, é apresentada no apêndice B. Nesta seção são apresentados os módulos mais importantes, que refletem as principais ações do *Time Warp* e a interconexão entre eles. São apresentadas também as estruturas dos principais tipos de *tokens* utilizados, para facilitar a compreensão das informações que são manipuladas pelo modelo.

Como simplificação em relação ao protocolo, o modelo básico desenvolvido não representa o cálculo de *GVT* e a operação de *fossil collection*. Optou-se por essa abstração para construir uma versão inicial simplificada do modelo do *Time Warp* onde considera-se que a utilização de algoritmos de cálculo de *GVT* não interfere na simulação distribuída (Srinivasan; Reynolds 1995b) e o fato de que espaço em memória é um recurso disponível em grande escala nas arquiteturas atuais. Assim como as demais características de implementação e de limitação do otimismo, diferentes algoritmos de cálculo de *GVT* e também de *fossil collection* podem ser introduzidos no modelo básico através dos *plugins* correspondentes.

A figura 4.6 apresenta os atributos dos principais *tokens* utilizados no modelo em Redes de Petri, representados por lugares que funcionam como depósitos ou estruturas de dados, abrangendo a *Input_Queue* (que armazena os eventos que deverão ser executados), a *Output_Queue* (que armazena a versão negativa de cada mensagem que o processo lógico envia), o *Buffer_de_Entrada* (que contém todas as mensagens e antimensagens recebidas da rede de comunicação), o *Buffer_de_Saída* (que armazena as mensagens e antimensagens que ainda aguardam para serem transmitidas pela rede de comunicação), a *Input_Queue_Executados* (eventos que são retirados da *Input_Queue*, executados e armazenados nessa fila para possibilitar o correto funcionamento do mecanismo de *rollback*) e *Ant_Queue* (que armazena as antimensagens que o processo lógico recebe). Além desses *tokens*, todos os demais lugares da Rede de Petri manipulam *tokens* que contêm o valor do *LVT* e outras informações de controle da simulação. O apêndice B mostra os algoritmos utilizados no manuseio dos *tokens* e a descrição das estruturas de forma mais detalhada.

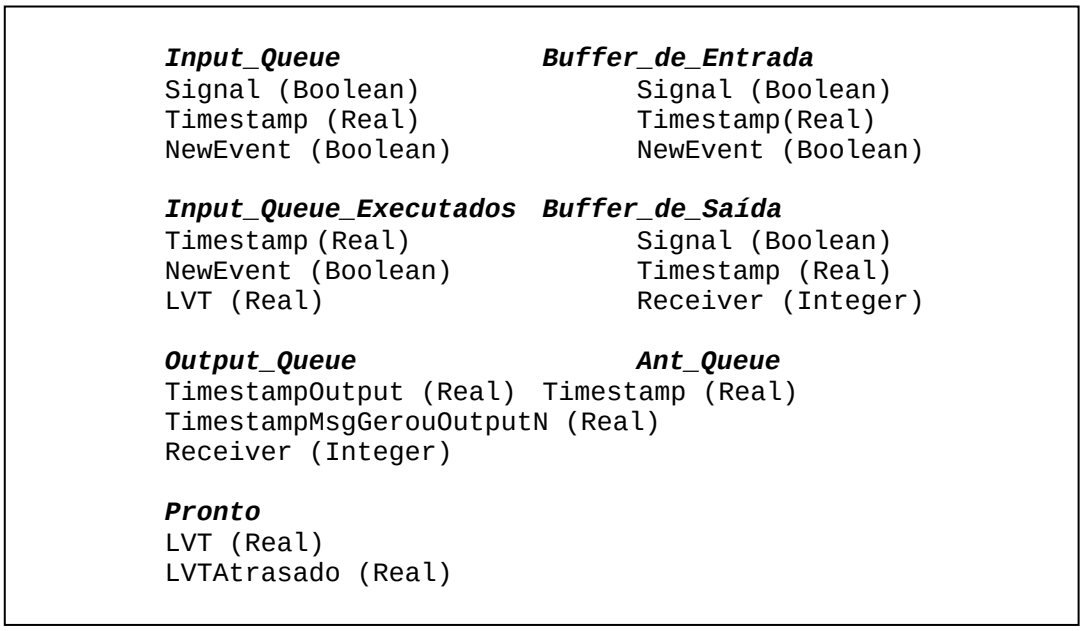


Figura 4.6: Lugares utilizados como Estruturas de Dados e Atributos dos *Tokens*.

A utilização das Redes Hierárquicas torna mais simples a apresentação do modelo, uma vez que o mesmo é bastante complexo. A figura 4.7 apresenta o modelo de um processo lógico *Time Warp* composto pelas super-páginas *Executa_Evento*, *Recebe_Msg/AntMsg* e *Envia_Msg/AntMsg*, correspondendo ao algoritmo de alto nível apresentado na figura 4.2.

O lugar *Pronto* reflete o estado do processo lógico apto a executar suas atividades de execução de um evento ou tratamento de uma mensagem ou antimensagem que está armazenada no *Buffer_de_Entrada*. Esse estado funciona como uma permissão que implementa a exclusão mútua entre essas atividades.

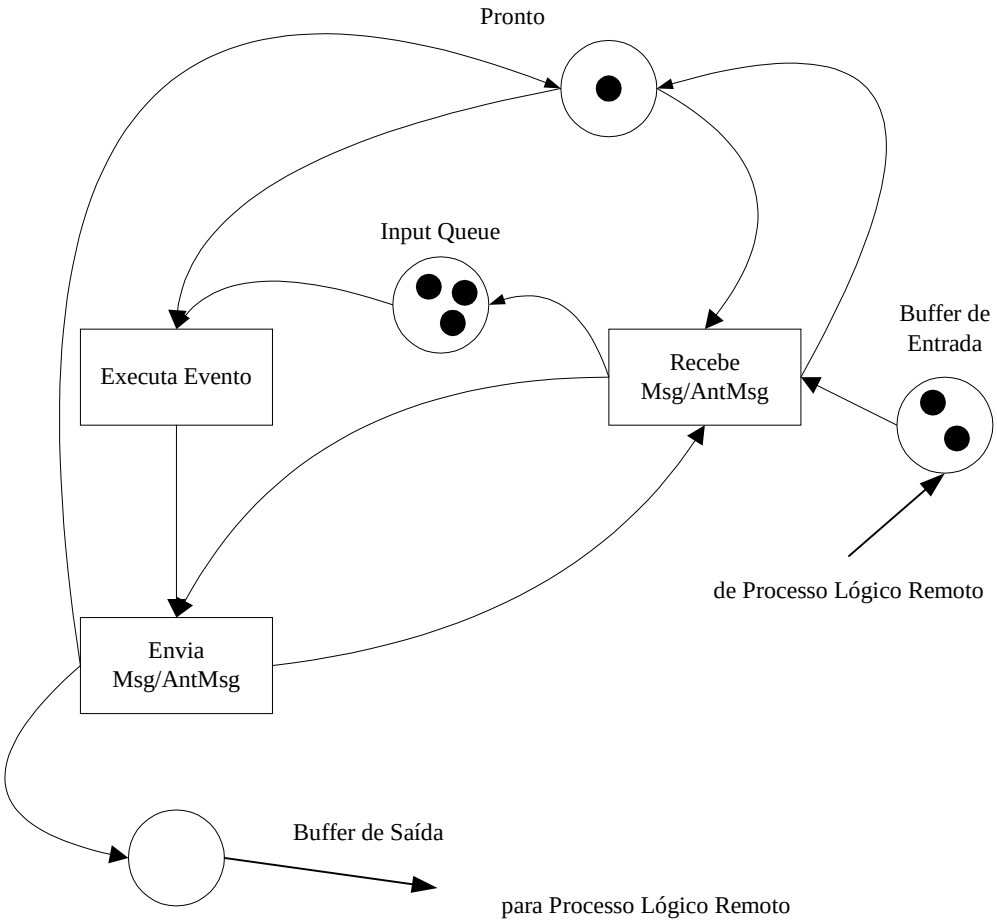


Figura 4.7: Modelo do Processo Lógico da Figura 4.2.

A figura 4.8 apresenta o modelo em Redes de Petri representando a execução de um evento que é retirado da *Input_Queue* (refinamento da super-página *Executa_Evento* da figura 4.7), mostrado no algoritmo da figura 4.3. A *Input_Queue* armazena *tokens*, que representam os eventos que serão executados, em ordem não decrescente da marca de tempo.

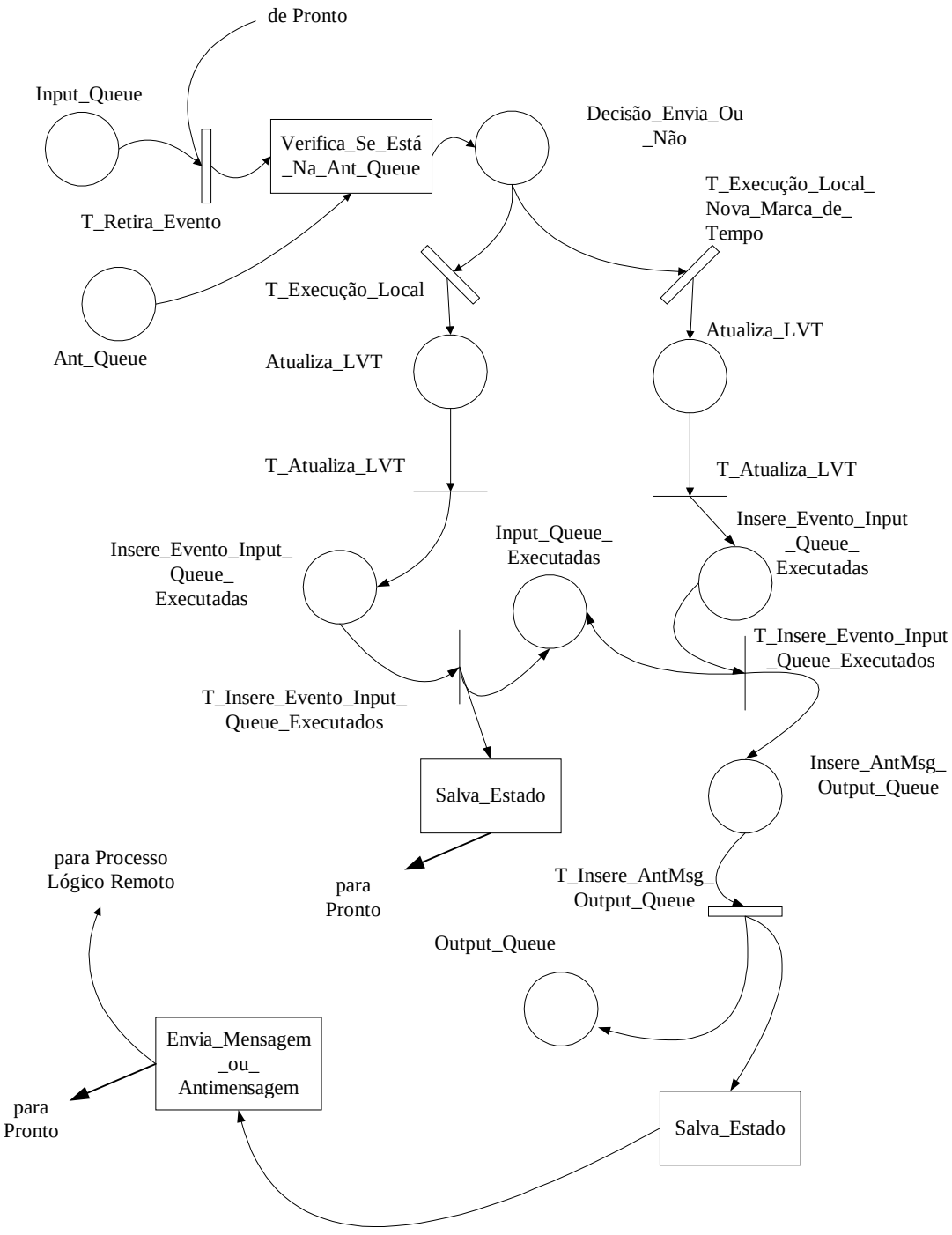


Figura 4.8: Modelo em Redes de Petri da Figura 4.3.

A representação das transições segue a definição apresentada no capítulo 2, diferenciando aquelas que são temporizadas das não-temporizadas. A parte referente ao salvamento de estados é apresentada através de uma ótica mais abstrata, com o uso de uma super-página, devido ao fato de que a sua representação em Redes de Petri pode variar, dependendo do algoritmo de salvamento utilizado.

O evento é retirado da *Input_Queue* e inicia-se uma busca pela *AntQueue*. Pode haver uma antimensagem que irá anulá-lo (super página *Verifica_Se_Está_na_AntQueue*).

O lugar *Decisão_Envia_ou_Não* representa a decisão de enviar um novo evento para um processo lógico remoto ou para a *Input_Queue* local e depende da aplicação. A transição *T_Retira_Evento* é responsável por obter o primeiro evento (com menor marca de tempo) da *Input_Queue*. As transições *T_Execução_Local* e *T_Execução_Local_Nova_Marca_de_Tempo* refletem a execução do evento e a diferença entre elas é que a segunda calcula a marca de tempo do novo evento gerado, com base no *LVT* do processo lógico e no tempo de serviço da aplicação. As transições *T_Atualiza_LVT* são imediatas porque o tempo necessário para atribuir o valor da marca de tempo do evento que acabou de ser executado ao valor de *LVT* pode ser desprezado em relação aos outros tempos gastos (como por exemplo a execução de evento e o salvamento de estados). As transições *T_Insere_Evento_Input_Queue_Executados* e *T_Insere_AntMsg_Output_Queue* também refletem o tempo gasto para manipular as estruturas de dados *Input_Queue_Executados* e *Output_Queue*.

A figura 4.9 mostra o refinamento da super-página *Salva_Estado*, representando o algoritmo *Copy State Saving*. Essa operação utiliza tempo para ser executada e por isso a transição correspondente é temporizada (*T_Salva_Estado*). Não é necessário utilizar um lugar representando a *State Queue*, uma vez que o modelo está interessado somente nas ações que afetam o desempenho, isto é, no tempo gasto para efetuá-las.

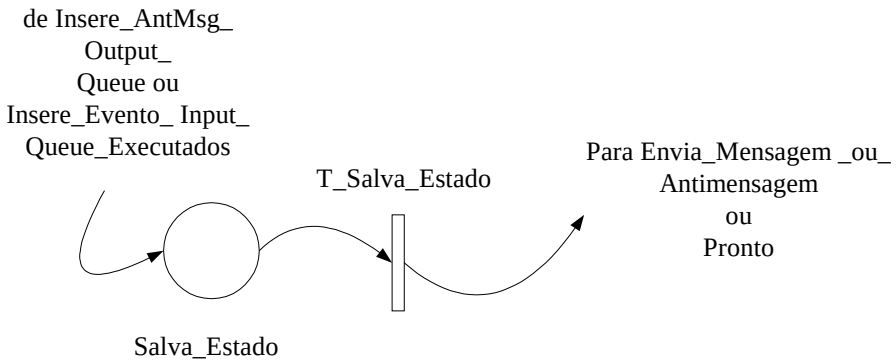


Figura 4.9: Refinamento da Super-página *Salva_Estado* Representando o Algoritmo *Copy State Saving*.

A figura 4.10 apresenta o refinamento da super-página *Envia_Mensagem_Ou_Antimensagem*. Após salvar o estado, o processo lógico insere uma cópia negativa da mensagem enviada na *Output_Queue* (transição temporizada *T_Atualiza_Output_Queue*), insere a mensagem no *Buffer_de_Saída* (*T_Insere_Buffer_de_Saída*) e retorna para o estado (lugar) *Pronto*, para receber ou executar novas mensagens de evento. Do *Buffer_de_Saída* as mensagens (ou antimensagens, em caso de ocorrência de *rollback*) são enviadas para os *Buffers_de_Entrada* dos processos destino, através das transições temporizadas *T_Envia*. Existem tantas transições *T_Envia* quantos forem os processos lógicos destinos, e essas transições utilizam como parâmetro o tempo de comunicação entre os processos lógicos. O lugar *Permissão_Para_Enviar* garante que uma única mensagem ou antimensagem seja enviada por vez.

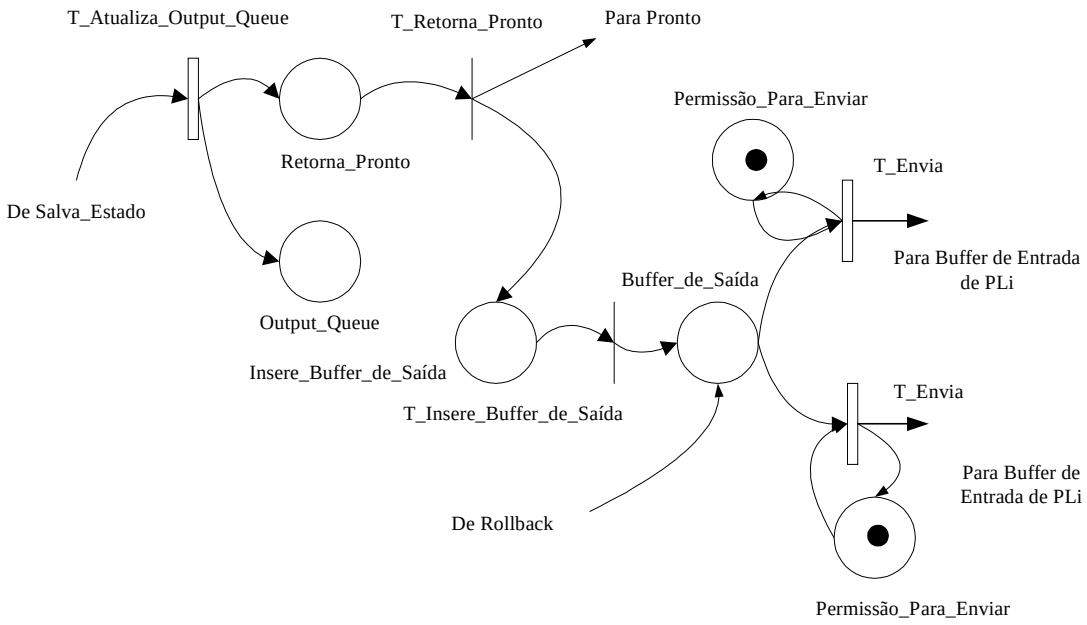


Figura 4.10: Refinamento da Super-Página Envia_Msg/AntMsg.

A figura 4.11 apresenta o refinamento da super-página *Recebe_Msg/AntMsg*. O processo lógico retira o próximo elemento do *Buffer de Entrada* e se for uma mensagem, verifica se a antimensagem correspondente está na *Ant_Queue*. Em caso afirmativo ambas são descartadas e um *token* é inserido no lugar *Pronto*. Senão, é feita a análise da marca de tempo da mensagem em relação ao *LVT* (lugar *Verifica_Marca_de Tempo*) e tomadas as providências em caso de *rollback* ou simplemente o evento é inserido na *Input_Queue* (transição temporizada *T_Insere_Evento_na_Input_Queue*).

Se o próximo elemento do *Buffer de Entrada* for uma antimensagem o processo lógico busca na *Input_Queue_Executados* o evento correspondente. Se não foi executado, a antimensagem é inserida na *Ant_Queue* e no momento da execução essa fila é analisada para verificar se o evento deve ser executado ou descartado. Se o evento foi executado um *rollback* é iniciado.

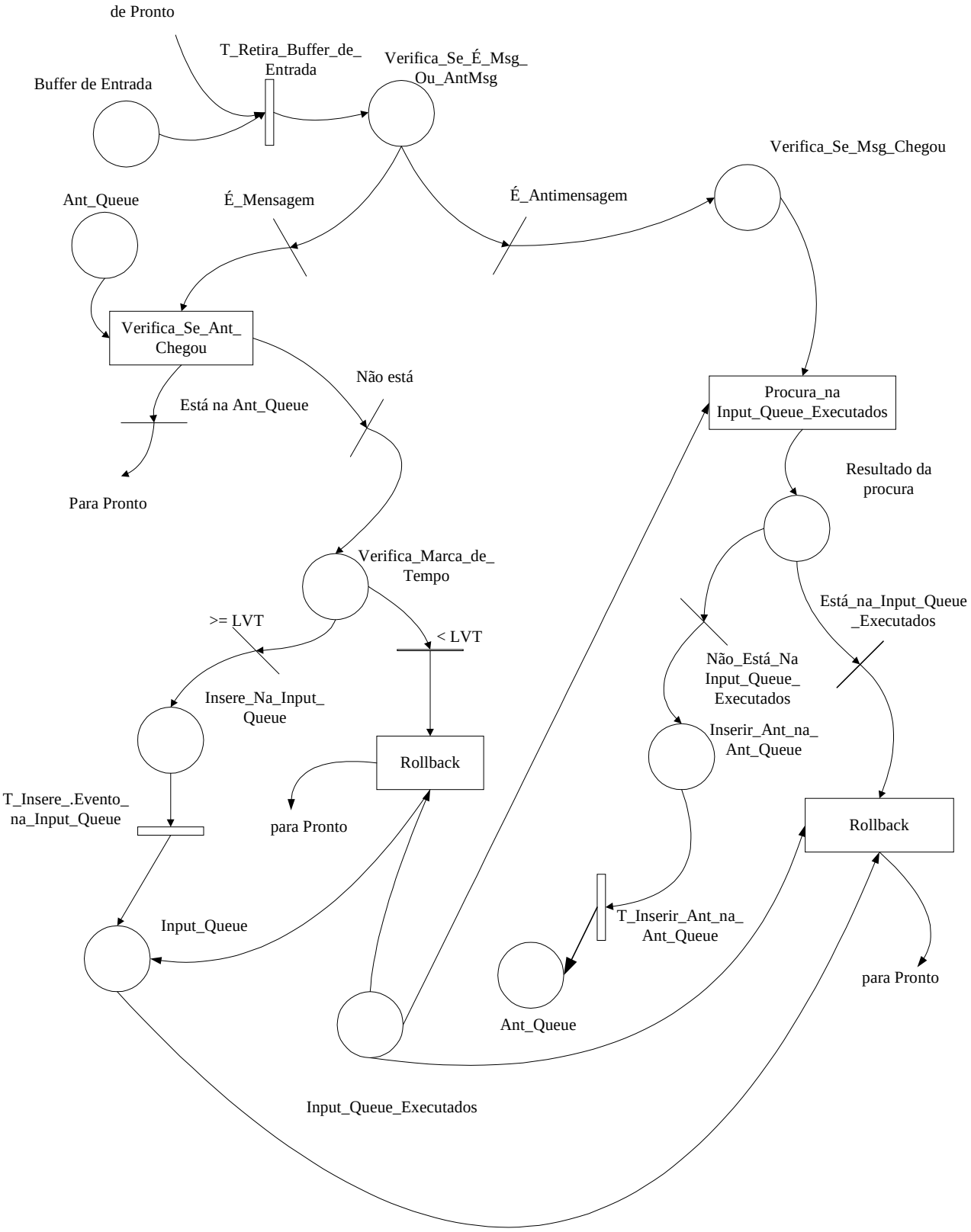


Figura 4.11: Refinamento da Super-Página *Recebe_Msg/AntMsg*.

A figura 4.12 mostra o refinamento da super-página *Rollback*.

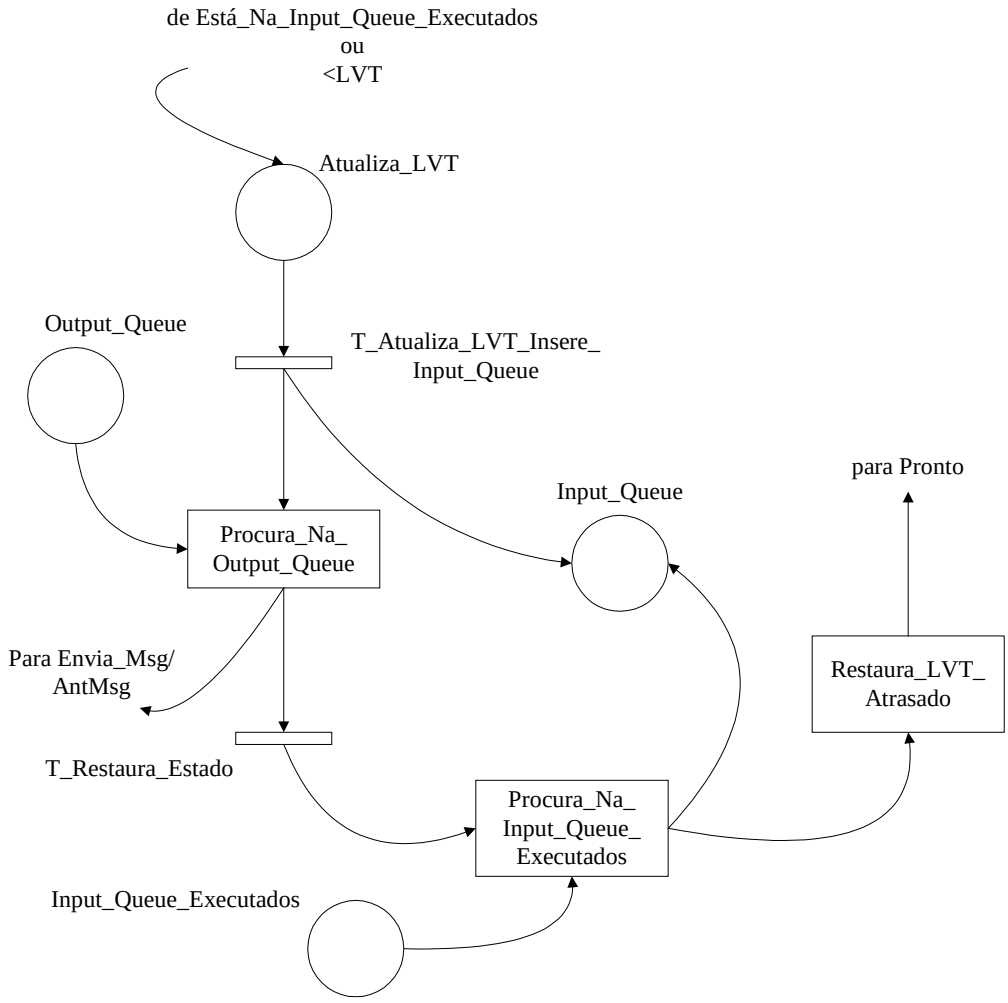


Figura 4.12: Refinamento da Super-Página *Rollback*.

A primeira tarefa efetuada no *rollback* é a atualização do *LVT* (se for *rollback* por mensagem) e a inserção do evento correspondente na *Input_Queue* (transição *T_Atualiza_LVT_Insere_Input_Queue*). Essa transição é temporizada porque leva em consideração o tempo gasto na manipulação da estrutura de dados *Input_Queue*.

O Cancelamento Agressivo consiste no próximo passo, que busca na *Output_Queue* todas as mensagens cujo envio foi provocado por eventos com marca de tempo maior do que a marca de tempo da mensagem que provocou o *rollback*. Imediatamente, as antimensagens correspondentes são enviadas para *Envia_Msg/AntMsg*, que as insere no *Buffer de Saída*.

O tempo para restaurar estado é representado na transição *T_Restaurar_Estado*. Após isso, todos os eventos com marca de tempo maior do que a marca de tempo do *rollback* são retirados da *Input_Queue_Executados* e inseridos

na *Input_Queue*, para serem reexecutados na ordem cronológica correta. *Restaura_LVT_Atrasado* consiste em busca o valor do parâmetro *LVT_Atrasado* referente ao último evento executado corretamente.

A diferença do *rollback* mostrado na figura 4.12 para o *rollback* provocado por antimensagem consiste no fato de que a transição *T_Atualiza_LVT_Insere_Input_Queue* passa a ser não temporizada, pois não há evento para inserir na *Input_Queue*. A atualização do *LVT* ocorre na super-página *Restaura_LVT_Atrasado*. O novo valor do *LVT* corresponde ao valor da marca de tempo do último evento executado corretamente antes da marca de tempo da antimensagem que provocou o *rollback*.

4.4.5 Verificação e Validação do Modelo

A correspondência entre o programa de simulação e o modelo (verificação) pode ser feita quase que diretamente analisando-se a especificação gráfica das Redes de Petri (a especificação gráfica reflete a lógica dos algoritmos apresentados na seção 4.4) e a construção dos *tokens* que circulam pela rede carregando as informações necessárias (por exemplo, o valor de *LVT* e a marca de tempo das mensagens e antimensagens).

A validação do modelo representativo do comportamento de um processo lógico *Time Warp* foi efetuada por meio de duas abordagens. A primeira através de cuidadosa comparação do modelo com as descrições da seção 4.4.2. Na segunda abordagem utilizou-se a técnica de *face validity*, na qual os resultados obtidos com a simulação são comparados com o comportamento esperado do sistema. A proposta de tempo virtual de Jefferson (Jefferson 1985) prevê o conjunto de ações que um processo lógico deve realizar em resposta a situações durante a simulação distribuída. O modelo foi submetido a esse conjunto de situações e o comportamento observado valida o modelo. A tabela 4.2 relaciona o conjunto de situações e o comportamento esperado.

Tabela 4.2: Validação do Modelo Representativo do Comportamento de um Processo Lógico *Time Warp*.

Situação	Comportamento Esperado
Chegada de uma mensagem e a antimensagem correspondente está armazenada na *Ant_Queue*	Ambas são descartadas
Chegada de uma mensagem e a antimensagem correspondente não está armazenada na *Ant_Queue*	O processo lógico verifica a marca de tempo da mensagem
Marca de tempo da mensagem que chegou é maior ou igual ao *LVT*	O evento correspondente é inserido na *Input Queue* para posterior execução
Marca de tempo da mensagem que chegou é menor do que o *LVT* do processo lógico	O processo lógico imediatamente ativa o procedimento de *rollback*
Chegada de uma antimensagem	O processo lógico procura na *Input Queue Executados* se o evento correspondente já foi executado
O evento correspondente à mensagem que a antimensagem deveria anular já foi executado	O processo lógico imediatamente ativa o procedimento de *rollback*
O evento correspondente à mensagem que a antimensagem deveria anular não foi executado ou não chegou	A antimensagem é inserida na *Ant_Queue*
Processo lógico ativa o *rollback* provocado por mensagem atrasada	O evento correspondente à mensagem é inserido na *Input Queue* e o processo envia antimensagens para todas as mensagens enviadas, referentes a eventos executados com marca de tempo maior do que a marca de tempo da mensagem atrasada
Processo lógico ativa o *rollback* provocado por antimensagem	O processo envia antimensagens para todas as mensagens enviadas referentes a eventos executados com marca de tempo maior ou igual do que a marca de tempo da antimensagem
Rollback	Todos os eventos executados acima do valor da marca de tempo da mensagem atrasada ou da anitmensagem voltam para a *Input Queue* para serem re-executados
Input Queue vazia	O processo lógico trata as mensagens e/ou antimensagens do *Buffer* de Entrada, se existirem
Buffer de entrada vazio	O processo lógico executa os eventos da *Input Queue*, se existirem

4.5 Desenvolvimento de *Plugins* para o Modelo Básico

Um *plugin* tem por objetivo, depois de anexado ao modelo básico, alterar o seu comportamento. A taxonomia proposta no capítulo 3 identificou, por exemplo, várias formas de se limitar o otimismo e várias formas de se efetuar salvamento de estados. A seleção por uma forma ou outra de limitação de otimismo e salvamento de estados é que vai conduzir a uma ou outra variante do *Time Warp*. Por exemplo, se a limitação do otimismo for nula e o salvamento de estados for do tipo *Copy State Saving*, tem-se o *Time Warp* básico. Por outro lado, com a simples substituição da limitação do otimismo de nula para probabilística, o modelo passa a representar o comportamento de um protocolo probabilístico, tal como *PDSP*.

O uso de modelos (um modelo básico *Time Warp* e modelos de protocolos variantes através do uso de *plugins*) permitirá o estudo dos protocolos em diferentes situações tais como, por exemplo, variações na arquitetura de hardware e software sobre a qual a simulação distribuída sincronizada pelo protocolo otimista deverá executar, caso seja implementada. A figura 4.13 apresenta o relacionamento entre o modelo básico, que representa um núcleo comum, e o conjunto de *plugins*. A partir dos pontos de conexão, podem-se anexar os *plugins* ao modelo básico e com isso obter variantes.

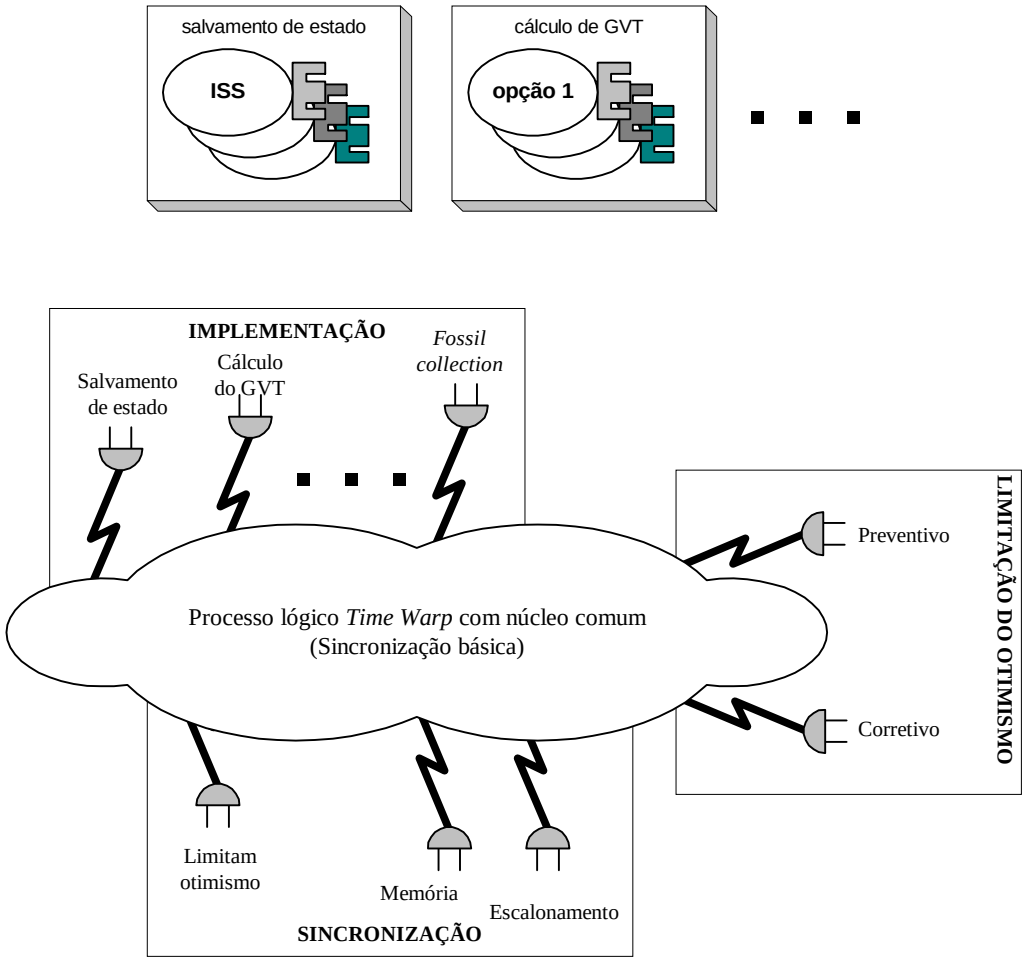


Figura 4.13: Relacionamento entre o Modelo Básico e os Domínios de *Plugins*.

Dessa forma, a taxonomia proposta identificou o domínio dos *plugins* e as variantes admitidas em cada domínio. Por exemplo, no domínio de Salvamento de Estados, as variantes admitidas são *Copy state saving*, *Sparse State saving* e *Incremental state saving*. Todas as variantes de *Time Warp* implementadas e que utilizam salvamento de estados utilizam uma das variantes especificadas acima. O mesmo vale para os demais domínios de *plugin*.

A figura 4.14 mostra a especificação de um *plugin* para o salvamento de estado *Sparse state saving* fixo. O lugar *Decide_Se_Salva_Estado* pode tomar a decisão com base, por exemplo, no número de eventos executados (a cada n eventos) ou no tempo de simulação (intervalos). Para a implementação da versão adaptativa, pode-se substituir esse lugar por uma super-página que detalha a tarefa de tomada de decisão. As transições *T_Salva_Estado* e

T_Não_É_Intervalo_de_Salvamento_de_Estado refletem os tempos gastos com o salvamento do estado do processo lógico e com a execução do algoritmo. Outro exemplo de *plugin* para o salvamento de estados foi apresentado na figura 4.9, que descreve o algoritmo *Copy state saving*.

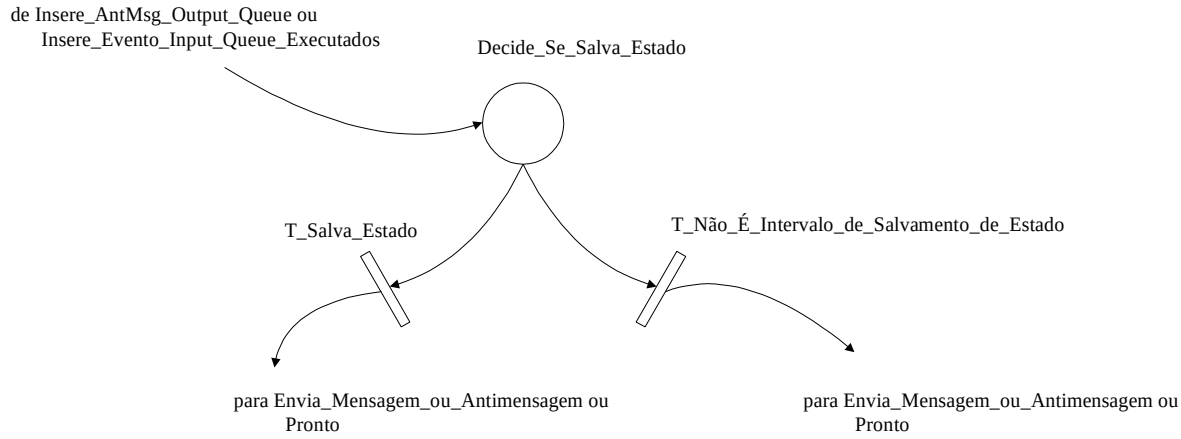


Figura 4.14: *Plugin Sparse state saving*.

Outro *plugin* é mostrado na figura 4.15, descrevendo o algoritmo que modifica o modelo básico para representar um processo lógico do protocolo de sincronização probabilístico *PDSP*. Após a tomada de decisão sobre o bloqueio ou não do processo lógico (lugar *P_PDSP*), o algoritmo permite que o processo prossiga para a execução normal de um evento (transição *T_Normal_Exec*) ou então bloqueia o processo lógico (*T_PDSP*). O evento é reinserido na *Input_Queue* para posterior execução e o processo volta ao estado Pronto. As transições *T_Normal_Exec* e *T_PDSP* são temporizadas para refletirem os tempos que o protocolo gasta com a execução do algoritmo e com o bloqueio do processo lógico.

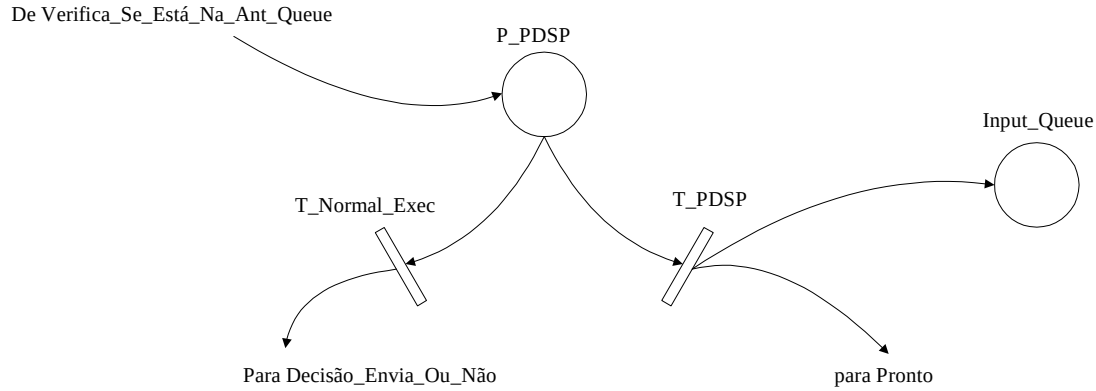


Figura 4.15: *Plugin* Protocolo *PDSP*.

4.6 Considerações Finais

Neste capítulo foram apresentadas a estrutura detalhada de um processo lógico *Time Warp* e considerações sobre o desenvolvimento e validação de um modelo refletindo o comportamento com base na estrutura apresentada. Foram utilizadas as Redes de Petri para representar o modelo, com solução através de simulação. O desenvolvimento do modelo foi facilitado pela taxonomia apresentada no capítulo 3 (identificação dos domínios de *plugins*) e pela descrição textual detalhada efetuada neste capítulo.

A motivação para este capítulo reside no fato de que a avaliação de desempenho de protocolos de sincronização em simulação distribuída é uma tarefa complexa e deve considerar as diferentes estratégias de implementação, otimização e as plataformas de hardware e ambientes de software utilizados. Existe uma carência de ferramentas práticas de avaliação dos diversos protocolos que vêm sendo apresentados. Atualmente, muitos autores utilizam casos e situações de teste específicos em detrimento de uma maneira mais clara, simples e abrangente de analisar desempenho.

As técnicas de modelagem são indicadas para efetuar essa análise exatamente pela não necessidade da presença física do objeto de estudo. A vantagem da simulação sobre a análise analítica reside no fato de que as alterações, que por ventura são impostas ao modelo, podem ser mais facilmente refletidas.

Dessa forma, optou-se por realizar o estudo de desempenho de protocolo *Time Warp* utilizando técnicas de modelagem, e a resolução do modelo por simulação. As grandes vantagens da simulação em relação às outras técnicas de avaliação são a possibilidade de representar a execução da simulação distribuída em diferentes plataformas e a capacidade de, através de pequenas alterações no comportamento do modelo de um processo lógico, representar processos lógicos de protocolos variantes do *Time Warp*.

Este capítulo apresenta algumas contribuições relevantes para a área de simulação distribuída, destacando-se:

- A identificação das atividades de um processo lógico *Time Warp* que são relevantes em uma análise de desempenho e a apresentação da

especificação dessas atividades em forma textual e em forma de modelo através das facilidades de representação gráfica que as Redes de Petri oferecem;

- A descrição detalhada, textual, do comportamento de um processo lógico *Time Warp* produzida na fase de confecção do modelo, e que pode ser utilizada como base para futuras implementações de protocolos otimistas;
- A definição do uso de domínios de *plugins* e sua utilização como extensões para alterar o modelo básico proposto.

O capítulo 5 descreve um método que utiliza o modelo apresentado e os domínios de *plugins* para efetuar a avaliação de desempenho de protocolos otimistas (*Time Warp* e variantes). Esse método considera também a influência da plataforma e da aplicação que será resolvida no desempenho da simulação distribuída.

Capítulo 5

Um Método para Avaliação de Desempenho de Protocolos de Sincronização Otimistas *Time Warp* e Variantes

Este capítulo descreve um novo método para avaliação de desempenho do protocolo de sincronização para simulação distribuída *Time Warp* e suas variantes, através do uso de simulação. O modelo básico descrito no capítulo 4 é utilizado como alicerce do método, ao qual *plugins* podem ser anexados. O usuário do método enxerga os processos lógicos como sendo caixas-pretas, sem a necessidde de trabalhar diretamente com a especificação em Redes de Petri apresentada no capítulo 4.

5.1 Considerações Iniciais

Este capítulo apresenta um novo método para avaliar o desempenho do protocolo de sincronização *Time Warp* e protocolos variantes, fornecendo aos usuários da simulação distribuída uma forma de escolher o protocolo de sincronização mais apropriado. Esse novo método utiliza a simulação para avaliar o comportamento do protocolo de sincronização.

O capítulo 4 apresentou o desenvolvimento, verificação e validação de um modelo básico do protocolo *Time Warp* (visão interna de um processo lógico) e a definição dos domínios de *plugins*, através dos quais modelos das variantes do *Time Warp* podem ser construídos. O modelo proposto consiste no núcleo do procedimento de avaliação que considera o modelo básico (representando o protocolo *Time Warp*) no qual as variantes (segundo a taxonomia proposta no

capítulo 3) podem ser facilmente introduzidas por meio de *plugins*, para testar diferentes opções de sincronização de programas de simulação distribuída otimista.

Os processos lógicos da simulação distribuída são vistos como caixas pretas que recebem e enviam mensagens e antimensagens, através de uma rede de comunicação. Essa construção é modular e hierárquica, porque a união dos processos lógicos e do conjunto de *plugins* será considerada como bloco básico. Para interligar os vários processos lógicos, é necessária alguma forma de comunicação entre eles. Isso conduz à criação de mais um domínio de *plugins*: o de comunicação. As mensagens carregam eventos que correspondem aos eventos da aplicação que está sendo resolvida pela simulação distribuída e as antimensagens são utilizadas pelo protocolo de sincronização para manter a relação de causa e efeito entre os eventos que são executados pelos processos lógicos.

Os processos lógicos interagem para que a simulação distribuída consiga resolver a aplicação, dessa forma isso também deve ser considerado na avaliação de desempenho. O modelo proposto no capítulo 4 consiste no núcleo do procedimento de avaliação que considera o modelo básico (representando o protocolo *Time Warp*), no qual as variantes (segundo a taxonomia proposta no capítulo 3) podem ser facilmente introduzidas por meio de *plugins*, para testar diferentes opções para a sincronização de programas de simulação distribuída otimista. A figura 5.1 mostra o ambiente da simulação, que considera vários processos lógicos e as informações relativas à aplicação e à rede de comunicação (Spolon et al. 2000a).

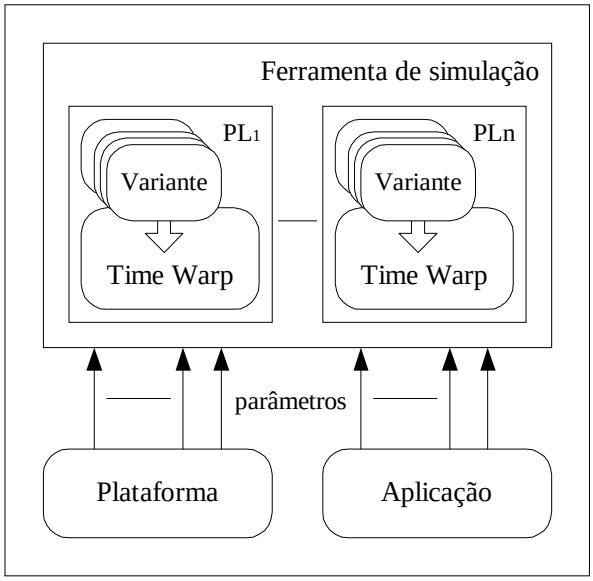


Figura 5.1: Ambiente de Avaliação de Desempenho de Protocolos de Sincronização Otimistas.

O conjunto de parâmetros necessários para instrumentar o modelo, para permitir a avaliação de desempenho do protocolo *Time Warp*, inclui os parâmetros relacionados ao protocolo de sincronização (isto é, o comportamento dos processos lógicos), à plataforma de interesse para implementar e executar a simulação distribuída e às características da aplicação descrita na simulação distribuída.

Os parâmetros relativos ao protocolo de sincronização estão encapsulados no modelo de um processo lógico mais *plugins*, representados como uma caixa preta. Os parâmetros relativos à plataforma são representados através de *plugins*. A aplicação deve refletir, no modelo do protocolo, os tempos entre-chegadas de clientes ou número de eventos iniciais, tempos de serviços, probabilidades nos pontos de decisão, o número de centros de serviço e a forma pela qual estão conectados (o fluxo de clientes pelo sistema). Esses valores de tempo são utilizados apenas para o cálculo das marcas de tempo das mensagens e não na temporização das transições.

Com relação à plataforma, os parâmetros necessários envolvem os tempos de comunicação entre os processos lógicos, tempo de execução de um evento, tempos para manipulação das estruturas de dados e tempo para salvar e restaurar estado (que depende do tamanho do estado a ser salvo). São esses valores de tempo que devem ser utilizados nas transições temporizadas do capítulo 4.

A seção 5.2 mostra a definição de um bloco básico que encapsula um processo lógico e a interconexão entre eles. A seção 5.3 discute a representação da plataforma no modelo do *Time Warp* básico. A seção 5.4 mostra exemplos de aplicação sendo representadas através da interconexão de blocos básicos. A seção 5.5 apresenta a validação do modelo utilizado no método de avaliação. Na seção 5.6 são apresentadas as considerações finais do capítulo.

5.2 Definição e Interconexão de um Bloco Básico

Esta tese propõe um método de avaliação que utiliza a simulação para avaliar os protocolos de sincronização otimistas. Essa simulação faz uso do modelo básico de um processo lógico *Time Warp* e os *plugins* (apresentados no capítulo 4), para representar a execução de uma simulação distribuída que resolve uma determinada aplicação e executa em uma certa arquitetura. Dessa forma, para construir a simulação do protocolo é necessário introduzir mecanismos que permitam a conexão entre vários modelos de processos lógicos e a representação da aplicação.

O método de avaliação de desempenho considera cada processo lógico *Time Warp* (PL) escolhido como um módulo, um bloco (figura 5.2) que envia e recebe mensagens e antimensagens. Esse bloco encapsula o modelo básico de um processo lógico e os *plugins* selecionados, apresentados no capítulo 4. Do ponto de vista do usuário do método de avaliação não interessa a forma com que a execução dos eventos e a sincronização acontecem dentro de cada um dos processos lógicos, mas sim o fato de que os processos lógicos se comunicam para garantir a execução e a sincronização da simulação distribuída.

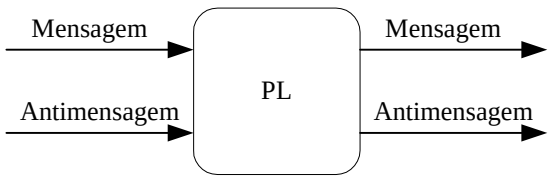


Figura 5.2: Processo Lógico Visto como um Bloco.

Cada processo lógico é responsável por executar os eventos de uma parte da aplicação, que pode ser um sistema aberto (os clientes chegam, recebem atendimento e deixam o sistema), fechado (existe um número fixo de clientes que circula pelo sistema) ou misto (uma parte dos clientes que chegam ao sistema vai embora e a outra parte retorna para receber atendimento). A figura 5.3 apresenta o bloco básico de um processo lógico que estende a funcionalidade do bloco da figura 5.2. Esse bloco deve ser utilizado pelo usuário do método de avaliação para representar a sua simulação distribuída, adaptado segundo as necessidades da aplicação (por exemplo, um sistema fechado não apresenta chegada ou saída de clientes, portanto as indicações do bloco básico Entrada e Saída devem ser omitidas (seção 5.4).

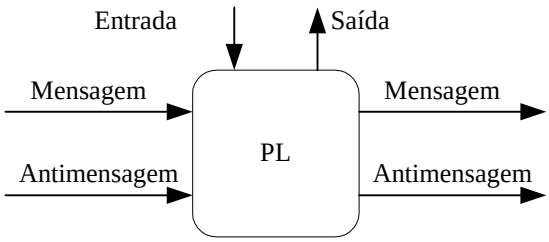


Figura 5.3: Bloco Básico de um Processo Lógico

A geração de clientes na aplicação, que alimenta a simulação distribuída, é representada pelo *plugin* da figura 5.4, que deve ser anexado ao processo lógico quando for necessário. Na figura 5.5 é apresentado o modelo em Redes de Petri da figura 5.4. A marca de tempo da mensagem que indica nova chegada de cliente ao sistema é calculada na transição *T_Nova_Chegada* (o valor de *LVT* mais o valor do tempo entre-chegadas).

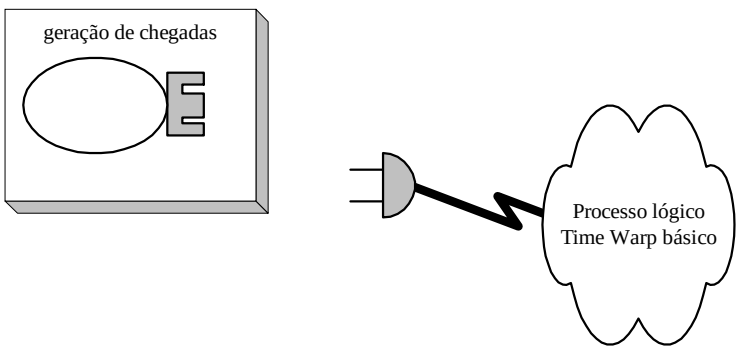


Figura 5.4: *Plugin* para Geração de Chegadas de Clientes.

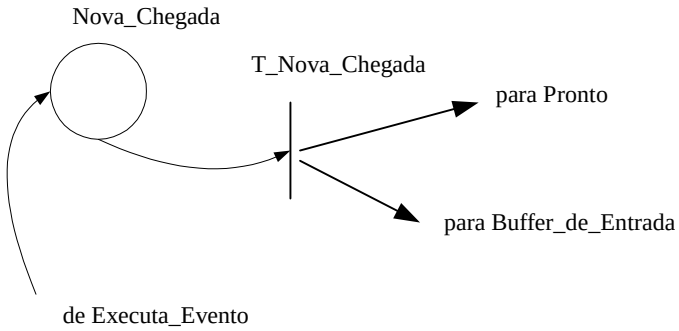


Figura 5.5: Modelo em Redes de Petri do *Plugin* Geração de Chegadas.

5.3 Representação das Características da Plataforma

O modelo para avaliar o protocolo de sincronização leva em consideração as características da plataforma, que envolvem a transmissão de mensagens e antimensagens, o processamento de eventos e as operações com as estruturas de listas.

As características do processamento são refletidas através da inserção de tempos nas transições que representam a execução de eventos, o salvamento de estados e a manipulação das estruturas de dados (as transições temporizadas foram apresentadas no capítulo 4).

Para construir a simulação do protocolo de sincronização que reflete as características de transmissão de mensagens e antimensagens considera-se que os processos lógicos são conectados através de uma rede de comunicação, como mostrado na figura 5.6.

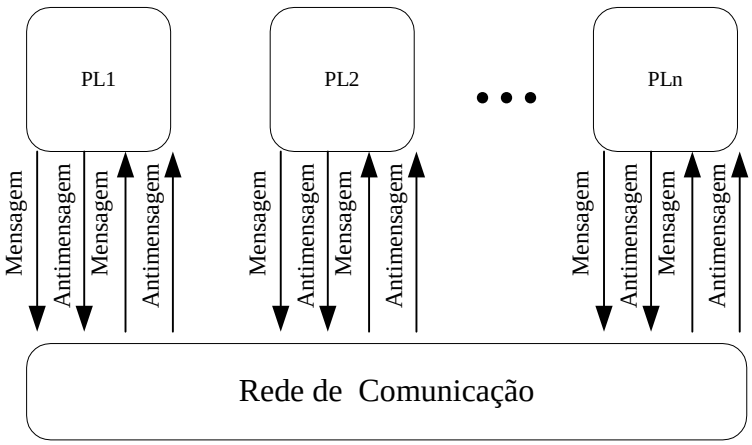


Figura 5.6: Processos Lógicos Conectados Através de uma Rede de Comunicação.

Estendendo as figuras 5.3 e 5.6 tem-se a figura 5.7, que mostra um bloco básico *Time Warp* e as entradas e saídas de mensagens e antimensagens que são recebidas e enviadas de/para a rede de comunicação. Apesar de compartilharem a mesma rede de comunicação, as mensagens e antimensagens são apresentadas em canais de entrada separados porque o tratamento que o processo lógico dispensa a elas é diferente, como discutido no capítulo 4.

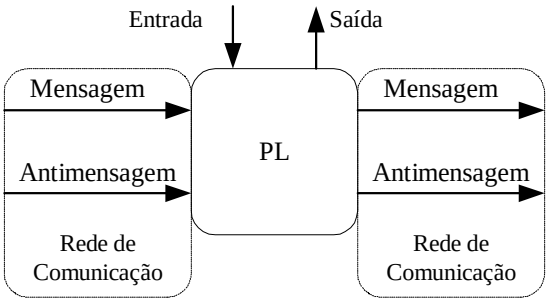


Figura 5.7: Bloco Básico Conectado a uma Rede de Comunicação.

Para manter a modularidade da simulação do protocolo de sincronização, a rede de comunicação também é considerada como um domínio de *plugins*, como definido no capítulo 4 (figura 5.8). Dessa forma, o modelo da rede de comunicação pode ser facilmente substituído para representar outras redes.

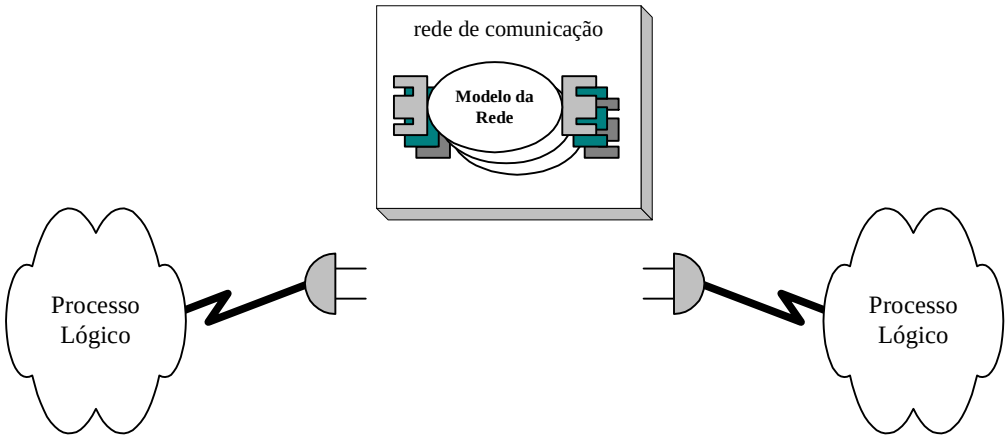


Figura 5.8: Processos Lógicos e o *Plugin* Rede de Comunicação.

Para exemplificar um modelo de rede de comunicação, pode-se considerar a Rede de Petri da figura 5.9, onde o meio físico não é compartilhado e, portanto, não

existe necessidade da espera para a transmissão (cada processo lógico deve receber um *plugin* de rede de comunicação). O tempo de transmissão é utilizado na transição *T_Envia* e o lugar *Permissão_Para_Enviar* garante que apenas uma mensagem ou antimensagem é enviada a cada transmissão.

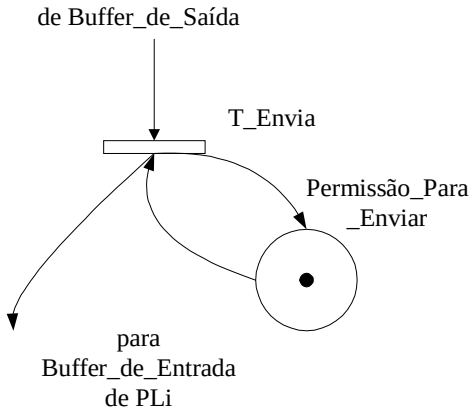


Figura 5.9: *Plugin* Representando Rede de Comunicação.

Plugins de redes de comunicação mais complexos podem ser inseridos, para permitir a análise da influência das características de diferentes meios de comunicação no desempenho da simulação distribuída.

5.4 Representação de Aplicações Utilizando os Blocos Básicos

Na seção 5.3 foi apresentado o bloco básico que deve ser utilizado para representar a simulação distribuída de uma aplicação, em uma determinada plataforma arquitetural. A maneira de interconectar os blocos básicos está relacionada com o particionamento da aplicação em processos lógicos.

Como exemplo pode-se considerar a aplicação da figura 5.10, que representa um modelo de filas de um sistema onde os clientes chegam com uma determinada taxa λ, são atendidos pelo Servidor 1 (tempo de atendimento μ_1) e depois dirigem-se ao Servidor 2 (tempo de atendimento μ_2). Após o término do atendimento, os clientes deixam o sistema.

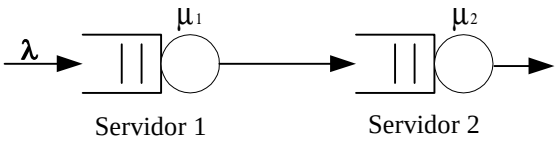


Figura 5.10: Aplicação Representando um Sistema de Filas em Série.

Para resolver essa aplicação utilizando-se simulação distribuída, a única configuração possível para dividí-la em processos lógicos é apresentada na figura 5.11. A representação através do uso de blocos básicos é apresentada na figura 5.12, que mostra a entrada dos clientes (no PL1) e a saída (no PL2) dos clientes da aplicação. A indicação de saída do PL_1 e de entrada no PL_2 foram suprimidas por não serem necessárias para essa aplicação em particular. As mensagens são enviadas em um único sentido, de PL1 para PL2 porque esse é o fluxo de clientes na aplicação.

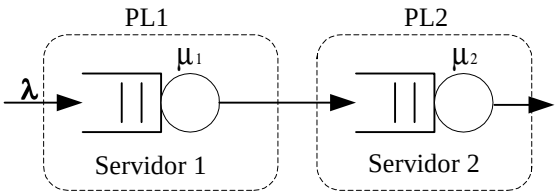


Figura 5.11: Divisão da Aplicação Filas em Série em Processos Lógicos.

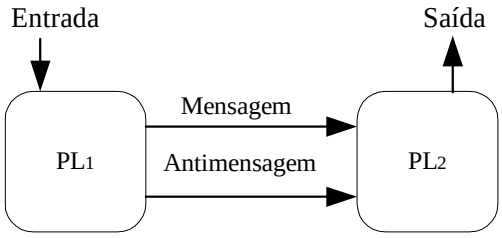


Figura 5.12: Representação da Simulação Distribuída da Aplicação Filas em Série Utilizando Blocos Básicos.

A figura 5.13 mostra o modelo de filas de um sistema computacional, utilizado por Ulson (Ulson 1999a, Ulson 1999b) como caso de teste para avaliação de desempenho do protocolo *CMB*. Nessa aplicação os clientes, ou tarefas do sistema, chegam e solicitam atendimento à UCP. Após isso, as tarefas podem deixar

o sistema, ou então solicitar o atendimento de uma das unidades de disco, Disco 1 ou Disco 2. Encerrado o atendimento no disco, a tarefa retorna para à UCP.

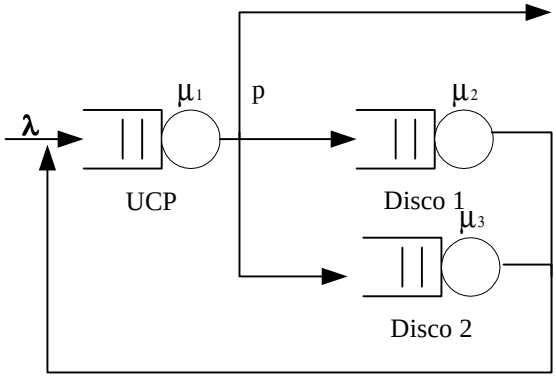


Figura 5.13: Sistema Computacional.

Uma configuração possível para dividir a aplicação em processos lógicos é apresentada na figura 5.14.

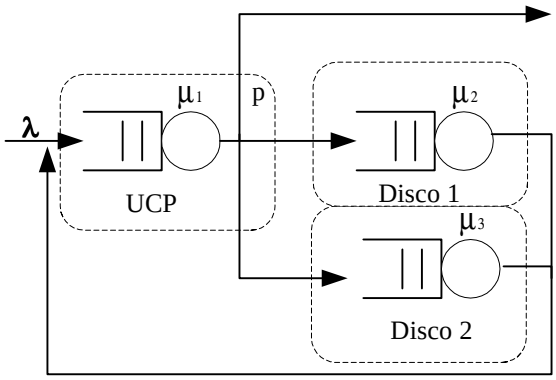


Figura 5.14: Divisão da Aplicação Sistema Computacional em Três Processos Lógicos.

A representação da aplicação, com a divisão apresentada na figura 5.14, utilizando os blocos básicos, é apresentada na figura 5.15. Os processos lógicos PL2 e PL3 não utilizam as representações de Entrada e Saída, somente o processo lógico PL1.

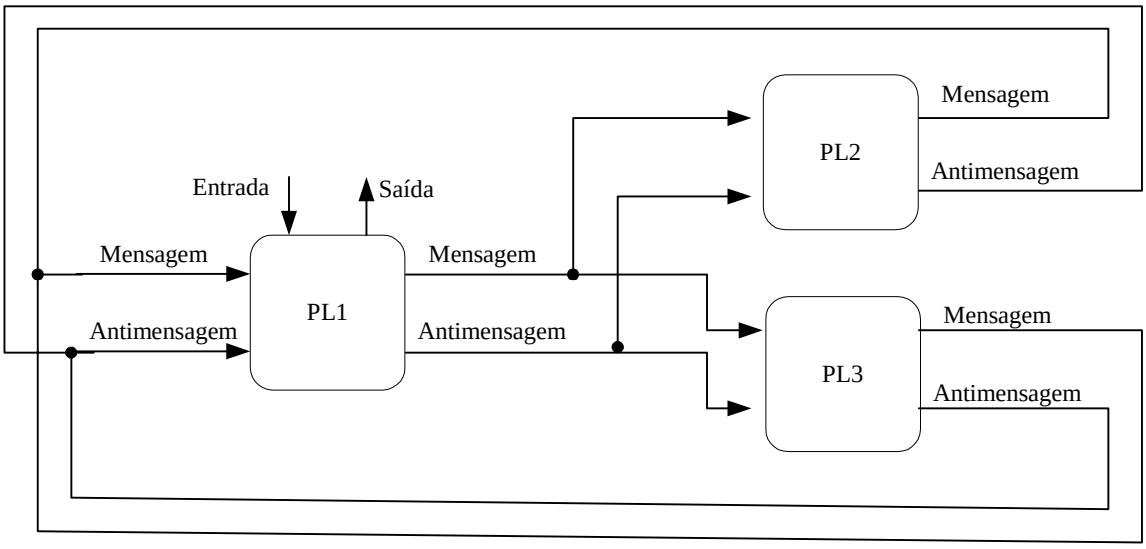


Figura 5.15: Representação da Simulação Distribuída da Aplicação Sistema Computacional Utilizando Blocos Básicos.

Até o momento discutiu-se o bloco básico que encapsula um processo lógico representando apenas um recurso da aplicação. Foi definido também um bloco básico que representa vários recursos (esse bloco consiste no processo lógico apresentado no capítulo 4, com pequenas modificações). Para exemplificar o seu uso, considera-se a aplicação da figura 5.13 dividida em dois processos lógicos (figura 5.16).

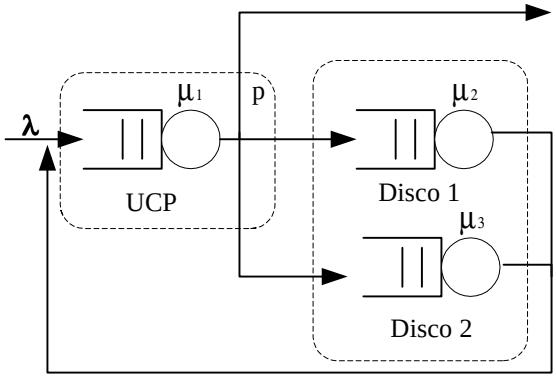


Figura 5.16: Divisão da Aplicação Sistema Computacional em Dois Processos Lógicos.

A figura 5.17 exemplifica a utilização de um bloco básico no qual o processo lógico encapsula mais de um recurso da aplicação. Nesse caso, as mensagens devem carregar a identificação do recurso destino.

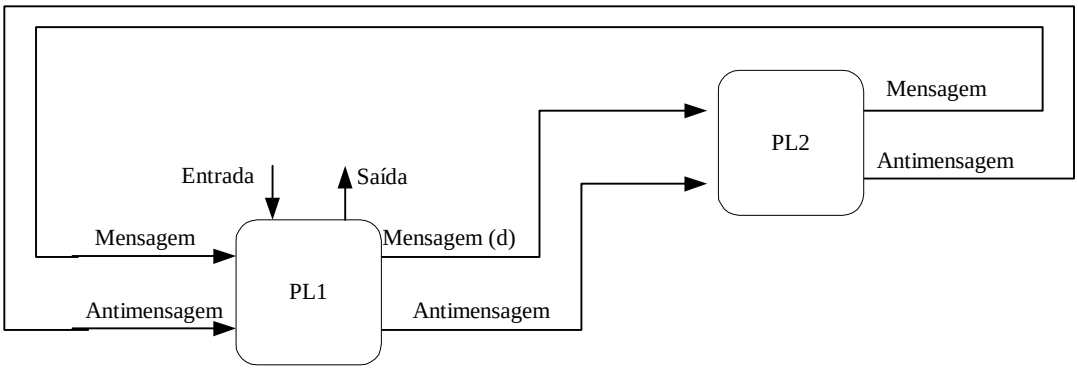


Figura 5.17: Representação da Simulação Distribuída da Aplicação Sistema Computacional Utilizando Blocos Básicos com Mais de um Recurso.

O desenvolvimento de um estudo de desempenho utilizando o método proposto pode ser descrito nas seguintes etapas:

1. Identificar a aplicação ou classe de aplicação (sistema aberto, fechado ou misto);
2. Identificar os parâmetros relativos à aplicação;
3. Dividir a aplicação em processos lógicos, utilizando os blocos básicos;
4. Identificar os parâmetros relativos à plataforma;
5. Configurar os parâmetros e anexar os *plugins* necessários;
6. Identificar as saídas desejadas e executar a simulação.

5.5 Verificação e Validação

As etapas de verificação e validação do ambiente que utiliza o modelo representativo do comportamento do protocolo *Time Warp* podem ser divididas em duas partes. A primeira parte refere-se à verificação e validação do modelo do comportamento do processo lógico (discutida no capítulo 4) e a segunda refere-se à validação da simulação, considerando a aplicação e a plataforma.

O método proposto utiliza a simulação de vários processos lógicos (que são visões macroscópicas do modelo básico apresentado no capítulo 4), portanto, a verificação e a validação do método correspondem à verificação e à validação da interação entre os processos lógicos da simulação distribuída, que resolvem uma aplicação e executam em uma determinada plataforma.

Além disso, a implementação pode ser verificada através dos resultados esperados, que foram obtidos com aplicações como descrito a seguir.

A aplicação da figura 5.10 apresentou um modelo de filas em série, o qual é utilizado para verificar o modelo do *Time Warp*. O modelo de filas em série não sofre *rollbacks*, porque nenhum dos Servidores possui realimentação ou tem mais de uma entrada de clientes. A divisão em dois processos lógicos mostra que os mesmos estão livres de erros de causa e efeito, porque nenhum deles recebe mensagens vindas de mais de um processo lógico, como abordado em (Rönngren et al 1996b).

A tabela 5.1 mostra o resultado da simulação otimista (usando o modelo básico do *Time Warp*) com os processos lógicos da figura 5.11. Os parâmetros utilizados relativos à aplicação não representam situações reais e foram definidos apenas para a verificação descrita nesta seção. Esses parâmetros envolvem o tempo entre-chegadas com valor 5,0 unidades de tempo, tempo de serviço do primeiro recurso igual a 10,0 unidades de tempo e tempo de serviço do segundo recurso igual a 20,0 unidades de tempo. Os parâmetros relativos à plataforma também não são representativos de sistemas reais e foram mantidos constantes e iguais a 1,0 unidade de tempo (porque o interesse reside em mostrar apenas que a ordem de execução dos eventos é respeitada, não importando a plataforma).

Tabela 5.1: Simulação da Aplicação da Figura 5.10.

Chegadas de clientes	***LVT* PL_1**	**Entrada em atendimento no Servidor 1**	**Liberação do Servidor 1**	***LVT* PL_2**	**Entrada em atendimento no Servidor 2**	**Liberação do Servidor 2**
0,0	0,0	0,0	10,0	10,0	10,0	30,0
5,0	5,0	10,0	20,0	20,0	30,0	50,0
10,0	10,0	20,0	30,0	30,0	50,0	70,0
15,0	15,0	30,0	40,0	40,0	70,0	90,0
20,0	20,0	40,0	50,0	50,0	90,0	110,0

A coluna Chegadas de Clientes mostra o tempo de chegadas dos clientes na aplicação. As colunas Entrada em atendimento no Servidor i apresentam o instante

de início de atendimento em cada servidor, na aplicação. As colunas *LVT* PL_i indicam a atualização do *LVT* de cada processo lógico, que ocorre no momento em que um evento é retirado da *Input Queue* para execução. As colunas Liberação do Servidor i indicam o instante em que o respectivo servidor é liberado pelo cliente, na aplicação. Os valores de tempo dos eventos correspondentes à aplicação não interferem no andamento da simulação distribuída, representado pelos valores dos *LVTs*. Analisando a tabela, verifica-se a chegada de cinco clientes ao sistema, nos instantes t=0,0, t=5,0, t=10,0, t=15,0 e t=20,0. Imediatamente o primeiro cliente é atendido pelo Servidor 1, enquanto o próximo aguarda sua vez de ser atendido. Quando o primeiro cliente deixa o Servidor 1, imediatamente é atendido pelo Servidor 2, que tem seu LVT atualizado. Com isso mantém-se a ordem correta de execução dos eventos da aplicação.

Outra aplicação utilizada para verificar o modelo do *Time Warp* foi o Servidor de Arquivos (Santana 1989b), apresentado na figura 5.18 e particionado em dois processos lógicos. Os clientes chegam ao Servidor, solicitam serviço à UCP e com probabilidade p deixam o sistema, ou então solicitam atendimento ao Disco com probabilidade 1 – p.

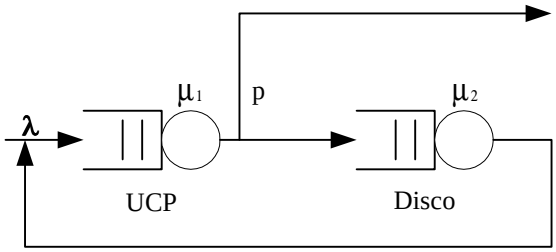


Figura 5.18: Servidor de Arquivos.

Os parâmetros utilizados para a execução da simulação não são representativos de um sistema real e foram definidos para a análise do comportamento da simulação distribuída que resolve essa aplicação.

Os parâmetros utilizados na aplicação foram tempo entre-chegadas igual a 5,0 unidades de tempo, tempos de serviço da UCP e Disco iguais a 10,0 e 20,0 unidades de tempo, respectivamente e probabilidade p igual a 0,7. Os parâmetros relativos à plataforma foram mantidos constantes e iguais a 1,0 (porque nesses experimentos o importante é verificar que a simulação distribuída executa os eventos na ordem

correta, não importando a plataforma). A tabela 5.2 apresenta os tempos entre-chegadas de 6 tarefas na aplicação e seu funcionamento até que todas as tarefas deixem o mesmo e os valores de LVT_1 e LVT_2. A ordem de execução das mensagens é preservada e a simulação do comportamento do *Time Warp* encerra quando a última mensagem é recebida pelo processo lógico LP_1, o evento correspondente é executado e não provoca a criação de outro evento, ou seja, a tarefa deixa o sistema.

Tabela 5.2: Simulação do Servidor de Arquivos com Chegada de Seis Tarefas.

Chegadas	***LVT* PL_1**	**Entrada em atendimento na UCP**	**Liberação da UCP**	***LVT* PL_2**	**Entrada em atendimento no Disco**	**Retorno para CPU**
0,0 *	0,0	0,0	10,0	0,0	---	---
5,0 *	5,0	10,0	20,0	20,0	20,0	40,0
10,0 *	10,0	20,0	30,0	30,0	40,0	60,0
15,0 *	15,0	30,0	40,0	30,0	---	---
20,0 *	20,0	40,0	50,0	50,0	60,0	80,0
25,0 *	25,0	50,0	60,0	60,0	80,0	100,0
40,0	40,0	60,0	70,0	60,0	---	---
60,0	60,0	70,0	80,0	80,0	100,0	120,0
80,0	80,0	80,0	90,0	90,0	120,0	140,0
100,0	100,0	100,0	110,0	110,0	140,0	160,0
120,0	120,0	120,0	130,0	130,0	160,0	180,0
140,0	140,0	140,0	150,0	150,0	180,0	200,0
160,0	160,0	160,0	170,0	170,0	200,0	220,0
180,0	180,0	180,0	190,0	190,0	220,0	240,0
200,0	200,0	200,0	210,0	190,0	---	---
220,0	220,0	220,0	230,0	190,0	----	---
240,0	240,0	240,0	250,0	250,0	250,0	270,0
270,0	270,0	270,0	280,0	280,0	280,0	300,0
300,0	300,0	300,0	310,0	310,0	310,0	330,0
330,0	330,0	330,0	340,0	310,0	---	---

A coluna Chegadas apresenta os eventos representando as chegadas de clientes na aplicação. As chegadas marcadas com * representam novos clientes chegando ao Servidor de Arquivos, enquanto as demais representam clientes que saem do Disco e retornam à UCP. As colunas Entrada em atendimento na UCP e Entrada em Atendimento no Disco apresentam o instante de início de atendimento em cada recurso, na aplicação. Quando essa última não apresenta nenhum valor, indica que o cliente deixou a aplicação. As colunas *LVT* PL_i indicam a atualização do *LVT* de cada processo lógico, que ocorre no momento em que um evento é retirado da *Input Queue* para execução. A coluna Liberação da UCP indica o

instante em que a UCP é liberada pelo cliente, na aplicação. A coluna Retorno para UCP indica o instante em que o cliente deixa o Disco e Retorna para a UCP. Os valores de tempo dos eventos correspondentes à aplicação não interferem no andamento da simulação distribuída, representado pelos valores dos *LVTs*.

Interpretando a tabela, verifica-se que no instante da aplicação t=0,0 um cliente chega ao sistema e é imediatamente atendido pela UCP. O *LVT* do PL_1 é atualizado para o valor 0,0. Em t=5,0 um novo cliente chega, o que provoca a atualização do *LVT* do PL_1 para o valor 5,0. Esse segundo cliente entra em atendimento na UCP em t=10,0 quando o primeiro cliente deixa a UCP e vai embora. Ele deixa a UCP em t=20,0 e imediatamente é atendido pela unidade de Disco (o *LVT* do PL_2 é atualizado para o valor 20,0), deixando a mesma em t=40,0.

Esse exemplo mostra que a atualização do *LVT* está correta, acontecendo quando um evento é executado e não sofre interferência da aplicação que está sendo considerada.

Outra aplicação utilizada para efetuar a verificação do modelo do comportamento do protocolo *Time Warp* foi o sistema computacional, apresentado na figura 5.13. Os parâmetros da aplicação utilizados (não representativos de um sistema real), todos exponencialmente distribuídos, foram: tempo entre-chegadas igual a 5,0 unidades de tempo, tempo de serviço na UCP igual a 10,0 unidades de tempo e tempo de serviço nas unidades de disco iguais a 20,0 unidades de tempo. O diagrama da figura 5.19 apresenta as marcas de tempo das mensagens trocadas entre os processos lógicos, onde PL1 representa a UCP, PL2 o Disco 1 e PL3 o Disco 2 (escolheu-se esse tipo de diagrama para apresentar esse caso porque simplifica a visualização da ocorrência do *rollback*). Ocorrem chegadas de clientes ao sistema com as marcas de tempo 0,0, 3,23, 7,11, 7,89 e 15,31. No PL1 o evento correspondente à mensagem com marca de tempo 0,0 é executado e gera uma mensagem para o PL2, com marca de tempo 8,05. O evento correspondente é executado e uma mensagem com marca de tempo 31,92 é enviada para PL1. Essa mensagem é executada na ordem correta (após as execuções dos eventos correspondente às chegadas) e não provoca a geração de uma nova mensagem, pois o cliente deixa a aplicação (probabilidade p).

A mensagem que o PL1 recebe do PL2, com marca de tempo 31,06 é recebida após a execução do evento correspondente à mensagem com marca de

tempo 61,10 vinda de PL3. Isso mostra a ocorrência de um *rollback* no Processo Lógico PL1. Depois de executar a mensagem com marca de tempo 61,10, o processo recebe a mensagem 35,06. Nesse caso, essa mensagem é inserida na *Input Queue* do processo para execução imediata e a mensagem 61,10, executada erroneamente, é reexecutada no momento oportuno. Nota-se que, nesse exemplo, essa execução errônea da mensagem de marca de tempo 61,10 não implicou em mensagem gerada para execução remota. Por isso, o *rollback* somente restaurou o estado de PL1 e não houve a necessidade de restaurar o estado dos demais processos lógicos.

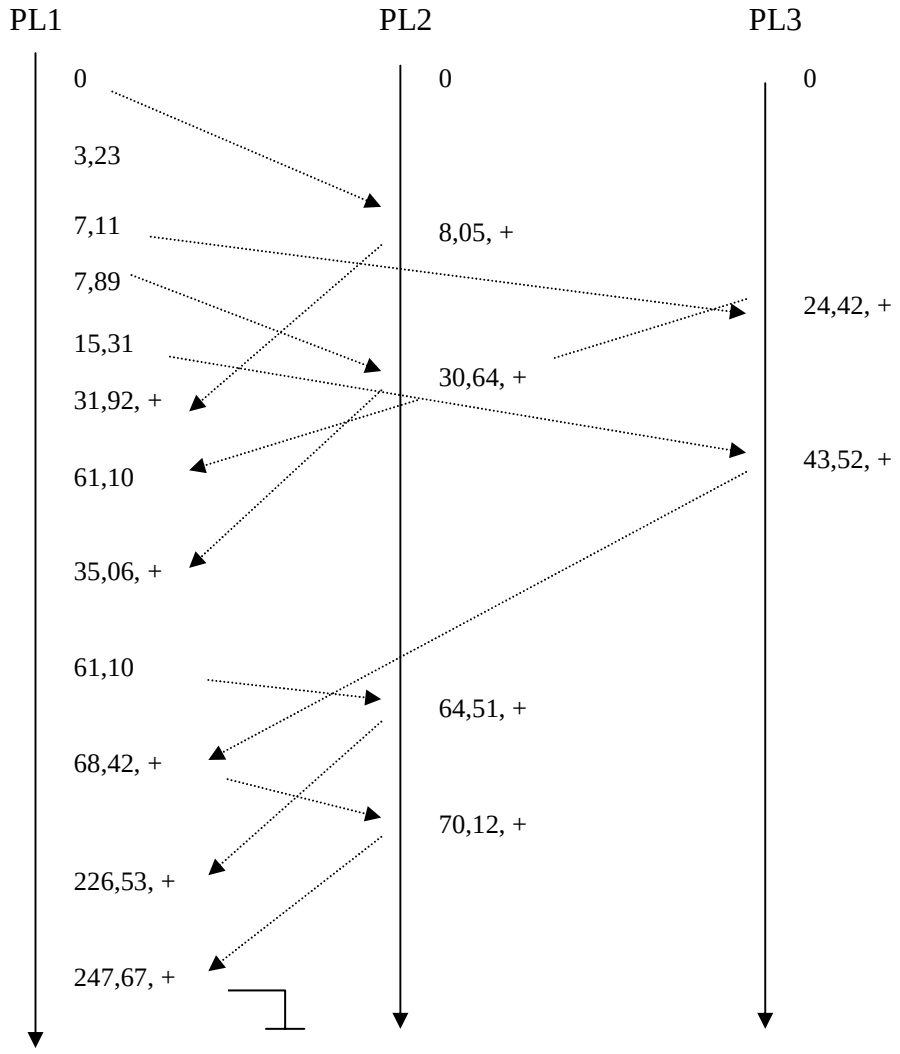


Figura 5.19: Simulação do Sistema Computacional com Chegada de 5 Tarefas.

Outras situações de teste também foram executadas, com o intuito de provocar a ocorrência de *rollbacks* e os resultados mostraram que a seqüência de execução dos eventos também foi mantida. O comprimento médio do *rollback* (isto é, o número médio de eventos desfeitos a cada ocorrência do *rollback*) manteve-se constante, em conformidade com resultados empíricos apresentados na literatura (Young; Wilsey 1996).

No capítulo 4 a tabela 4.2 apresenta a validação do comportamento do modelo de um processo lógico *Time Warp*. Quando se considera a interação entre esses processos essa validação se estende ao conjunto, uma vez que o mecanismo de envio de mensagens e antimensagens está validado, o mesmo acontecendo com o mecanismo de recepção. Todas as situações previstas, como por exemplo a chegada de uma mensagem atrasada, estão sendo abordadas na tabela 4.2. A comunicação entre os processos lógicos envolve o envio e recepção de mensagens e antimensagens, e a tabela mostra que o comportamento esperado acontece. Acrescenta-se ao conjunto de situações da tabela 4.2 mais duas: "Simulação resolve uma aplicação onde o número de chegadas se encerra" e **"**Todas as *Input Queue* e *Buffers* de entrada estão vazios (situação correspondente a quando os processos lógicos atualizam *LVT* para ∞)". O comportamento esperado para a primeira situação é "A simulação pára e todas as *Input Queue* e *Ant Queue* são esvaziadas e a simulação se encerra" e para a segunda situação é "A simulação se encerra".

5.6 Considerações Finais

Neste capítulo foi proposto um novo método para avaliação de desempenho dos protocolos de sincronização otimistas para simulação distribuída. O método acrescenta o domínio de *plugins* referente à comunicação entre os processos lógicos e às características da aplicação ao modelo básico de um processo lógico *Time Warp* e os domínios de *plugins* descritos no capítulo 4. A partir daí, o usuário do método enxerga cada processo lógico como uma caixa preta que envia e recebe mensagens e antimensagens, para resolver uma dada aplicação em uma determinada arquitetura. Dessa forma, o usuário não precisa trabalhar diretamente com a especificação em Redes de Petri de um processo lógico.

A descrição do comportamento de um processo lógico *Time Warp* foi apresentada no capítulo 4. Essa descrição, juntamente com a taxonomia proposta no capítulo 3, forneceu o embasamento para o desenvolvimento de um modelo básico. Esse modelo básico, juntamente com os parâmetros da aplicação e os parâmetros da plataforma, permite que uma simulação distribuída otimista seja simulada. Os parâmetros da plataforma envolvem o tempo de comunicação e tempo de processamento (para execução de evento, salvamento de estados, manipulação das estruturas de dados) e os parâmetros referentes à aplicação referem-se ao número e configuração dos servidores, tempos de serviço, tempo entre-chegadas se a aplicação representar um sistema aberto, número de eventos iniciais se o sistema for fechado e as probabilidades nos pontos de tomada de decisão. Um *plugin* gerador de chegadas foi apresentado, para ser anexado quando a aplicação representar um sistema aberto ou misto.

Este capítulo apresenta algumas contribuições importantes para a área de simulação distribuída, destacando-se:

- A proposta de um novo método de avaliação de desempenho dos protocolos de sincronização otimistas, que simula a simulação distribuída (meta-simulação) de forma hierárquica, através do uso de *plugins*;
- A identificação dos parâmetros necessários ao modelo do protocolo de sincronização (resultante da integração de processos lógicos componentes da simulação *Time Warp* que está sendo simulada), relativos à aplicação e à plataforma. Esses parâmetros podem ser inseridos no modelo através do ambiente de avaliação apresentado na figura 5.1.
- A praticidade do novo método, pois não é necessário implementar o protocolo, para se ter uma previsão do seu comportamento em uma determinada arquitetura (a complexidade do protocolo otimista *Time Warp* o torna um algoritmo difícil de implementar);
- A definição do *plugin* gerador de chegadas de clientes e o *plugin* para rede de comunicação.

O capítulo 6 descreve estudos de caso que mostram a viabilidade do método proposto e também fornecem alguns resultados de desempenho dos protocolos *Time Warp* e *PDSP*.

Capítulo 6

Avaliação dos Protocolos *Time Warp* Básico e *PDSP*

Este capítulo discute como podem ser obtidos resultados de desempenho utilizando-se o método de avaliação proposto no capítulo 5. São apresentadas as métricas que podem ser utilizadas e uma análise dos protocolos *Time Warp* básico e *PDSP* utilizando-se aplicações características de sistemas abertos, mistos e fechados.

6.1 Considerações Iniciais

Neste capítulo ilustra-se a aplicação do método de avaliação de desempenho proposto no capítulo 5. O protocolo *Time Warp* básico e o protocolo variante *PDSP* são utilizados como estudos de casos e demonstram a viabilidade de se utilizar o método para estudar o comportamento de protocolos otimistas. Além disso, os resultados de desempenho obtidos permitem comparar o *Time Warp* básico e o *PDSP* em diferentes situações de variações dos parâmetros e mostrar que o protocolo variante apresenta melhor resultado em relação ao protocolo básico. Foram realizadas as análises estatísticas necessárias (Apêndice D).

Diferentes aplicações são consideradas para a análise, envolvendo sistemas abertos, fechados e mistos. A seção 6.2 descreve essas aplicações e a interconexão dos blocos básicos *Time Warp* para os processos lógicos nos quais a aplicação é dividida. A seção 6.3 discute as métricas que podem ser utilizadas para verificar o comportamento dos protocolos de sincronização. A seção 6.4 discute o

comportamento do protocolo *Time Warp* no decorrer da simulação distribuída. Na seção 6.5 são discutidos os efeitos da influência da arquitetura em função da variação nos tempos de comunicação e processamento. A seção 6.6 estuda a influência da variação dos parâmetros da aplicação e a seção 6.7 discute a avaliação do protocolo probabilístico *PDSP* em relação ao *Time Warp* básico. A seção 6.8 discute como pode ser feita uma previsão aproximada do *speedup* da simulação distribuída em estudo, a partir das informações fornecidas pelo método de avaliação. A seção 6.9 apresenta as considerações finais do capítulo.

6.2 Aplicações Consideradas para a Análise

Nesta seção são descritas as aplicações utilizadas na análise do comportamento do protocolo *Time Warp* básico e do protocolo *PDSP* e a forma de dividir essas aplicações em processos lógicos. Constituídos os processos lógicos, os mesmos são representados através da interconexão dos blocos básicos *Time Warp*.

Uma vez que o objetivo principal dessa etapa do trabalho está concentrado em mostrar a viabilidade do uso do método de avaliação proposto no capítulo 5, optou-se por utilizar como estudo de caso aplicações que apresentam características comuns para as classes de sistemas abertos, fechados e mistos. Isso permite abranger uma grande variedade de situações, sendo consideradas diferentes parametrizações para cada uma das situações.

Define-se que um sistema é fechado, quando a quantidade de clientes no mesmo é constante e determinada antes do início da execução da simulação. Por outro lado, um sistema aberto contém um número de clientes variável, que depende da chegada de novos clientes e da saída de clientes do sistema. Um sistema misto apresenta clientes que podem retornar ao sistema, além dos novos clientes que chegam.

6.2.1 Aplicação 1: Servidor de Arquivos

A figura 6.1 mostra o exemplo de aplicação correspondendo a um sistema misto representando um Servidor de Arquivos, o que foi utilizado no capítulo 5 para fins de verificação.

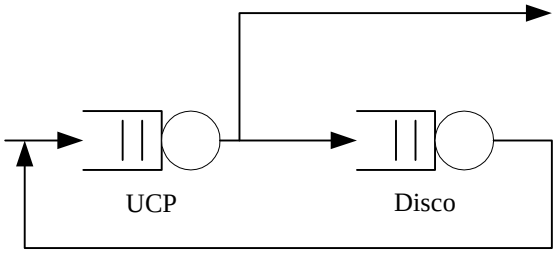


Figura 6.1: Aplicação 1: Modelo do Servidor de Arquivos.

A figura 6.2 apresenta a única configuração possível para dividir a aplicação, com apenas dois processos lógicos. Na figura 6.3 é apresentada a configuração dos blocos básicos *Time Warp* para os processos lógicos da figura 6.2.

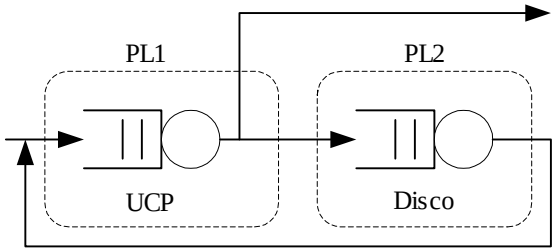


Figura 6.2: Aplicação 1 (Configuração).

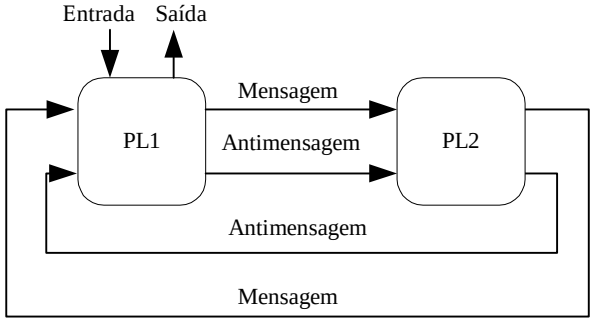


Figura 6.3: Configuração dos Blocos Básicos *Time Warp* para a Aplicação 1.

6.2.2 Aplicação 2: Servidor Central Simplificado

Essa aplicação representa um sistema computacional constituído por um elemento de processamento e duas unidades de disco e consiste em uma versão simplificada de uma aplicação utilizada na literatura nos estudos em simulação

seqüencial (MacDougall 1987) e também utilizado para avaliação da simulação distribuída (Fabbri 1999, Morselli 2000, Ulson 1999b), fatores esses que motivaram sua escolha.

Um número fixo de tarefas é escalonado para execução e permanece no sistema. Uma tarefa requisita a utilização da Unidade Central de Processamento (UCP), que atende a solicitação quando estiver disponível. Após o término do processamento, uma das unidades de disco (Disco 1 ou Disco 2) é selecionada aleatoriamente e a solicitação de atendimento é encaminhada para a unidade de disco correspondente. Encerrado o processamento na unidade de disco, uma nova requisição para utilizar a UCP é gerada e o ciclo de processamento se repete. A figura 6.4 apresenta o modelo referente a essa aplicação.

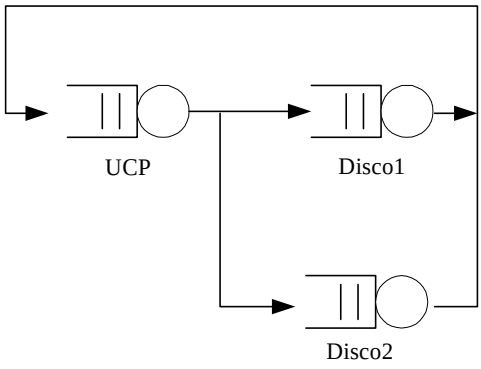


Figura 6.4: Aplicação 2: Modelo do Servidor Central Simplificado.

Os estudos de desempenho efetuados com essa aplicação e descritos nas seções seguintes consideram diferentes configurações para os processos lógicos que compõem a simulação distribuída. Utilizando-se três processos lógicos, a única maneira de agrupar os recursos da aplicação em processos lógicos é apresentada na figura 6.5. Na figura 6.6 é apresentada uma divisão em dois processos lógicos, agrupando os recursos que representam as unidades de disco em um único processo lógico. As figuras 6.7 e 6.8 mostram, respectivamente, como ficam as organizações dos blocos básicos *Time Warp* considerando as configurações das figuras 6.5 e 6.6.

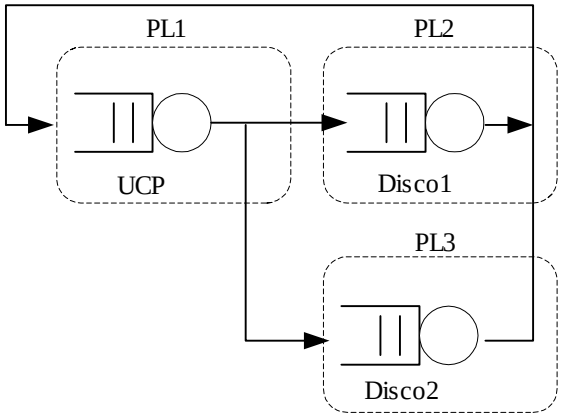


Figura 6.5: Aplicação 2 (Configuração 1).

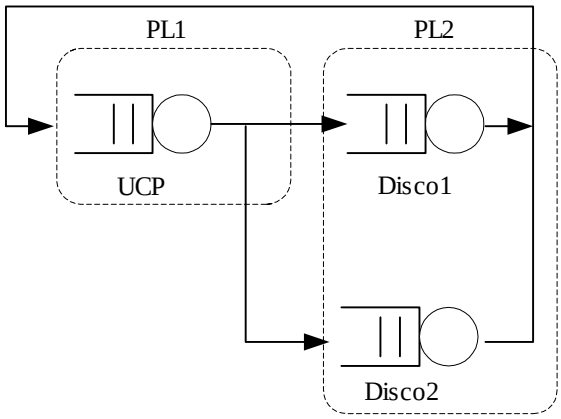


Figura 6.6: Aplicação 2 (Configuração 2).

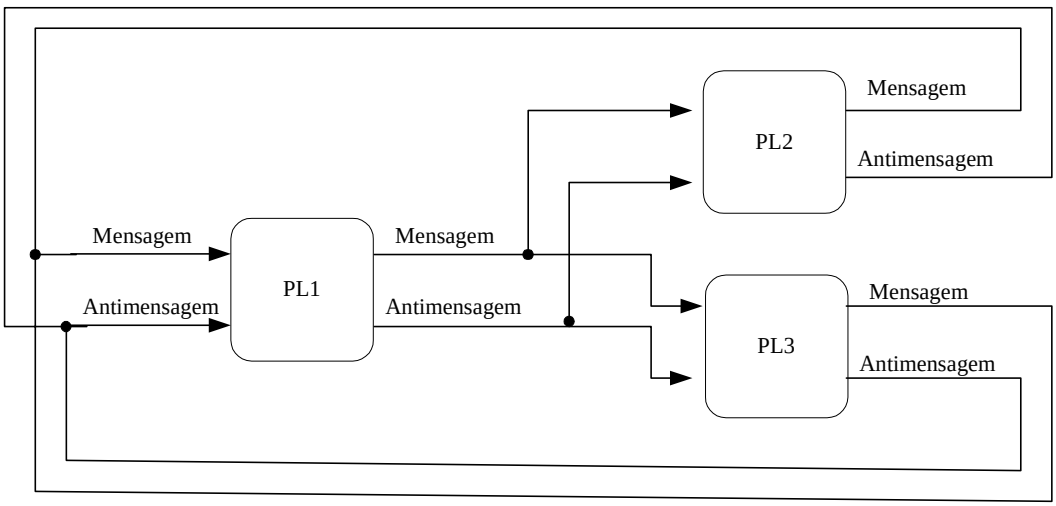


Figura 6.7: Configuração dos Blocos Básicos *Time Warp* para a Aplicação 2 (Configuração 1).

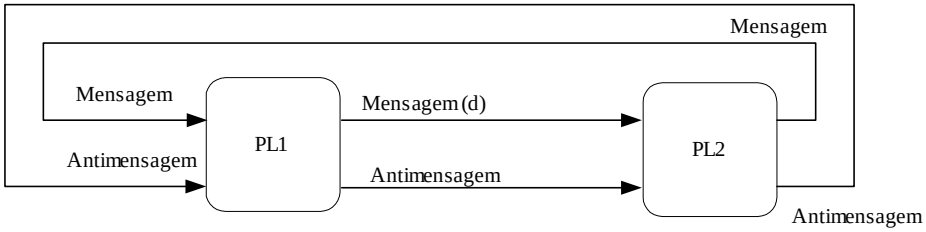


Figura 6.8: Configuração dos Blocos Básicos *Time Warp* para a Aplicação 2 (Configuração 2).

Como discutido no capítulo 4, um processo lógico *Time Warp* pode sofrer *rollback* quando recebe uma mensagem atrasada no tempo ou uma antimensagem, que surgem em decorrência de *rollbacks* provocados por uma mensagem atrasada ou por antimensagens. Dessa forma, uma mensagem atrasada no tempo é o ponto de partida para a existência de *rollbacks* e antimensagens na simulação distribuída otimista. Uma vez que na configuração 2 o processo lógico 2 recebe mensagens somente do processo lógico 1 e vice-versa, não existe maneira de um ou ambos receberem mensagens atrasadas no tempo. Portanto, essa configuração da Aplicação 2 não provoca a ocorrência de *rollbacks* na simulação distribuída e todos os eventos são executados na ordem correta.

6.2.3 Aplicação 3: Filas em Série

A aplicação 3 descreve um sistema de filas em série, utilizado no capítulo 5 (seção 5.4). Além da aplicação, foram apresentados o particionamento em processos lógicos e a configuração dos blocos básicos (seção 5.4). Como o processo lógico 1 somente recebe as mensagens relativas às chegadas de clientes e o processo lógico 2 recebe mensagens somente do processo lógico 1, não existe maneira de ambos receberem mensagens atrasadas no tempo. Portanto, como no caso da configuração 2 da Aplicação 2, essa aplicação não apresenta a ocorrência de *rollbacks* na simulação distribuída e todos os eventos são executados na ordem correta.

6.3 Parâmetros e Métricas Utilizados e Saídas Analisadas

Segundo Elmaghraby (Elmaghraby et al. 1999) as medidas de desempenho obtidas a partir de uma simulação distribuída otimista devem ter tratamento diferenciado, quando comparadas com as medidas obtidas de outras aplicações distribuídas. Como exemplo tem-se a utilização. Em um sistema computacional, a utilização corresponde à razão entre o tempo ocupado em relação ao tempo ocioso do processador. Já no *Time Warp* essa medida ou métrica (critério para comparar o desempenho (Jain, 1991)) deve ser adaptada, uma vez que nem todo o trabalho (ou os eventos executados) é correto, a utilização deve refletir apenas o tempo gasto com eventos executados na ordem correta.

Algumas das métricas utilizadas na literatura abrangem a freqüência de *rollbacks* (a razão entre o número de *rollbacks* e o número total de eventos executados), o comprimento médio de *rollback* (número médio de eventos desfeitos por cada ocorrência de *rollback*), a freqüência de antimensagens (número de antimensagens por unidade de tempo), a taxa de evento (número de eventos executados corretamente por unidade de tempo) (Fabri 1999, Cortellessa; Quaglia 2000), *rollbacks* por eventos (número de eventos que provocam a ocorrência de *rollbacks* em relação ao número de eventos executados corretamente (Tay et al. 1997), eficiência (porcentagem de eventos executados corretamente em relação ao total de eventos executados) (Ferscha; Johnson 1999).

As métricas, nesse caso, dizem respeito ao desempenho da simulação distribuída e diferem das informações estatísticas coletadas da aplicação (como por exemplo o comprimento médio da fila de um recurso e o tempo médio que um cliente aguarda por atendimento). Nos estudos apresentados neste capítulo as métricas utilizadas são a eficiência, comprimento médio do *rollback* e a freqüência do *rollback*. Segundo Ferscha (Ferscha; Johnson 1999) a eficiência é a melhor maneira de se analisar a simulação distribuída porque mostra o trabalho útil executado, e por essa razão essa métrica é utilizada neste trabalho. Fabbri (Fabbri 1999) justifica o uso do comprimento médio e a freqüência do *rollback*, entre outras métricas, por serem independentes do tempo da simulação.

As saídas analisadas permitem obter os valores das métricas, como por exemplo o número de eventos executados, o número de eventos executados corretamente, o número de *rollbacks*, o número total de eventos desfeitos.

Os parâmetros refletem as características da aplicação, da plataforma e também do protocolo de sincronização utilizado e são abordados nas subseções 6.3.1 e 6.3.2.

Para todas os valores de saída observados (total de trinta execuções para cada situação dos parâmetros de entrada) foram feitos os testes estatísticos adequados utilizando-se o cálculo do intervalo de confiança, *z-score* e testes de hipóteses (Mendenhall; Sincich 1992).

6.3.1 Parâmetros Relativos à Plataforma e ao Protocolo de Sincronização

Os parâmetros relativos à plataforma de execução da simulação distribuída podem ser divididos em dois grupos: referentes ao tempo de processamento e referentes ao tempo de comunicação (tabela 6.1).

Os parâmetros referentes ao tempo de processamento da plataforma foram obtidos através de coleta de dados, realizada por aferição, em uma estação de trabalho IBM Risk/6000 e utilizando o ambiente SMPLX (Ulson 1997). Esses parâmetros referem-se aos tempos médios de execução de um evento e à manipulação das estruturas de dados (*Input Queue*, *Output Queue*, *Ant Queue*), que incluem o salvamento e a restauração dos estados de um processo lógico.

O tempo médio de comunicação entre os processos lógicos foi estimado utilizando-se os resultados apresentados por Ulson (Ulson 1999a) (tabela 6.1). A partir desses parâmetros iniciais, os tempos de processamento e de comunicação foram variados para refletir outras arquiteturas.

Os parâmetros relativos ao protocolo de sincronização são definidos de acordo com o *plugin* utilizado, no caso o *plugin* referente ao protocolo *PDSP* que será analisado nas seções seguintes. Como visto no capítulo 4, são necessários o tempo de execução do algoritmo e o tempo de bloqueio do processo lógico (que estão relacionados com os parâmetros de processamento e, por isso, estão também relacionados na tabela 6.1). No decorrer do texto, os tempos de processamento e

comunicação definidos na tabela serão referenciados como padrão, a partir do quais outros valores serão definidos para representar plataformas diferentes.

Tabela 6.1: Parâmetros Relativos à Plataforma.

Processamento	
Parâmetro	**Valor (ms)**
Tempo médio de execução de um evento	1,7291
Tempo médio necessário para localizar uma mensagem/antimensagem nas estruturas de dados de um processo lógico	Equação 6.1
Tempo médio de inserção nas estruturas de dados	312
Tempo médio necessário para efetuar o salvamento do estado do processo lógico (o tamanho do estado a ser salvo é 1Kbyte, tamanho utilizado na literatura para permitir o estudo da influência do algoritmo de salvamento de estados no desempenho da simulação distribuída) (Rönngren et al. 1996a)	0.356
Tempo médio necessário para localizar um estado na State Queue e restaurar esse estado	Equação 6.2
Tempo médio para executar o algoritmo de avaliação do *PDSP*	O mesmo tempo de execução de um evento
Tempo de bloqueio de um processo lógico, utilizado no *plugin* PDSP	O mesmo tempo de execução de um evento
Comunicação	
Parâmetro	**Valor (ms)**
Tempo médio de comunicação	2,47

Equação 6.1:

0.005 + (n – 1) * (0.000000083681275 * Tamanho da estrutura + 0.00030466589) onde n corresponde ao n-ésimo elemento da estrutura de dados

Equação 6.2

0.035 + (n-1) * (-0.000000005653 * Tamanho da fila + 0.0007642) onde n corresponde ao número de eventos que foram desfeitos no *rollback* (no caso

de *rollback* causado por antimensagem utilizar-se n porque deve ser restaurado o último estado salvo antes da mensagem que foi retirada).

As equações 6.1 e 6.2 foram determinadas através de interpolação linear do conjunto de dados obtidos e construídas utilizando-se o programa Advanced Grapher (respectivamente, figura 6.9 e figura 6.10).

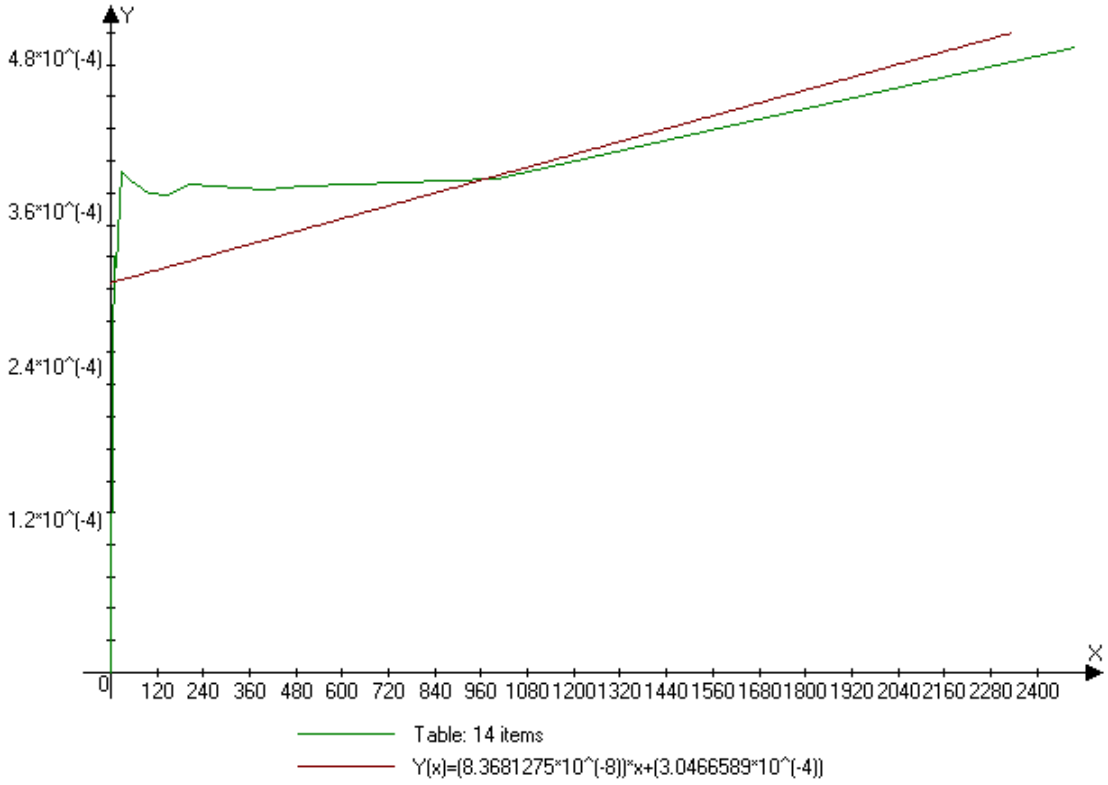


Figura 6.9: Obtenção da Equação 6.1 a Partir dos Tempos Obtidos.

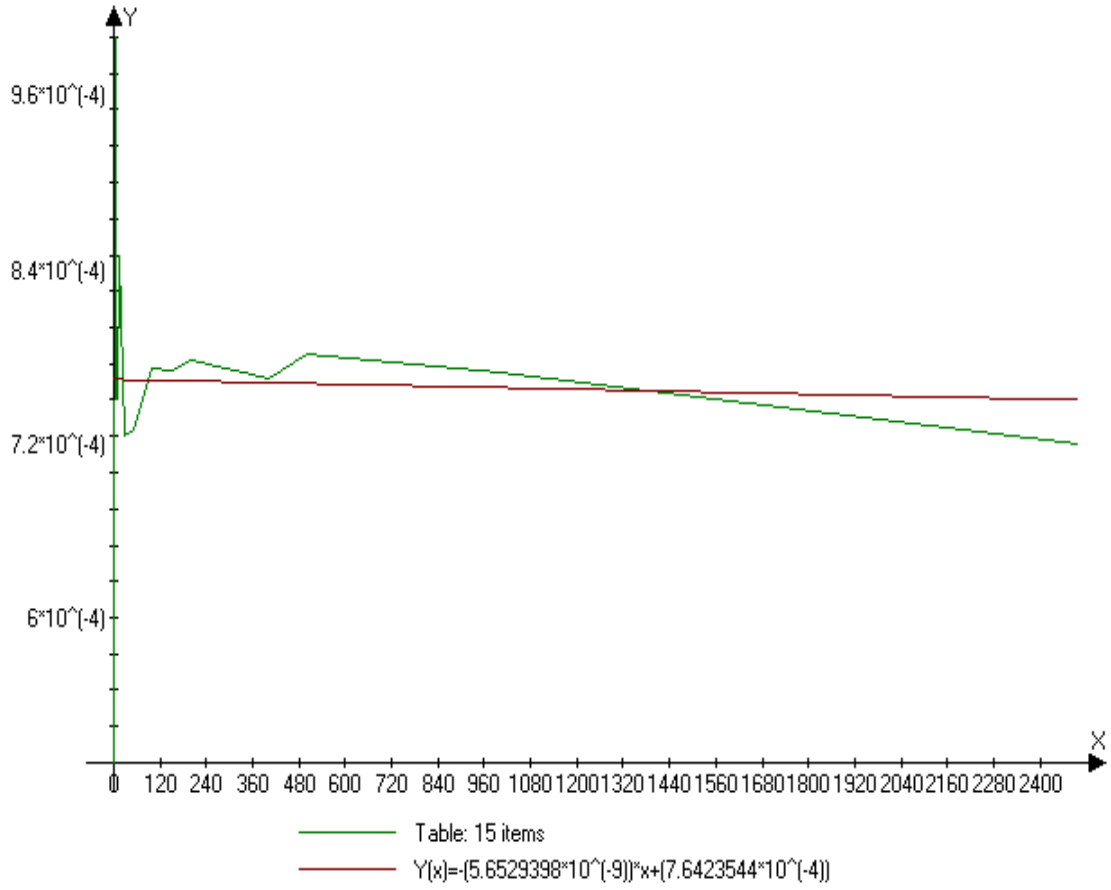


Figura 6.10: Obtenção da Equação 6.2 A Partir dos Tempos Aferidos.

No decorrer deste trabalho os valores dos parâmetros de tempo de processamento e tempo de comunicação apresentados nesta seção serão referenciados como padrão. Alterações desses valores serão utilizadas para representar plataformas diferentes.

6.3.2 Parâmetros Relativos à Aplicação

Cada aplicação possui seu próprio conjunto de dados de entrada, que servem como parâmetros do modelo que as representa. No modelo da Aplicação 1 são necessários a taxa λ de chegadas (ou o tempo entre-chegadas igual a 0,083 unidades de tempo) de novas tarefas, o tempo de serviço μ_1 da UCP (0,04 unidades de tempo), o tempo de serviço μ_2 (0,05 unidades de tempo) da unidade de Disco e a probabilidade p de uma tarefa deixar o sistema (0,6) (figura 6.11). Esses parâmetros

foram retirados do trabalho de Santana (Santana 1989b) e também utilizados por Silva (Silva 2000).

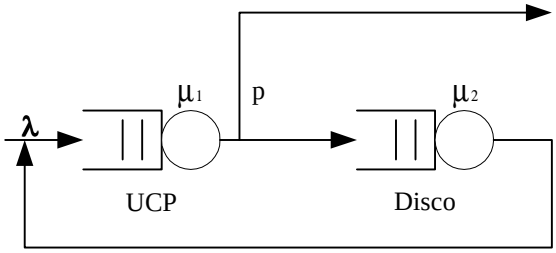


Figura 6.11: Parametrização da Aplicação 1.

Os parâmetros necessários ao modelo da Aplicação 2 referem-se ao número de tarefas inicialmente escalonadas na simulação (7 tarefas), além dos tempos de serviço μ_1 (10 unidades de tempo), μ_2 (20 unidades de tempo) e μ_3 (30 unidades de tempo) da UCP e das unidades de Disco. Assim como na aplicação Servidor de Arquivos, existe uma probabilidade p que indica se uma tarefa segue para a unidade de Disco 1 ou para a Unidade de Disco 2 (probabilidade igual para ambas unidades de disco) (figura 6.12). Esses parâmetros não são representativos de um sistema real e foram adaptados para a realização das análises, a partir dos parâmetros utilizados por MacDougall (MacDougall 1987).

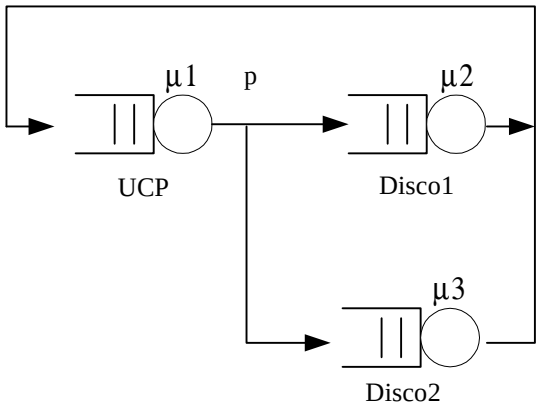


Figura 6.12: Parametrização da Aplicação 2.

6.4 Comportamento do Protocolo no Decorrer do Tempo da Simulação Distribuída

Os estudos efetuados com o protocolo *Time Warp* básico possibilitaram a verificação de seu comportamento no decorrer do tempo da simulação distribuída, em dois intervalos: o primeiro para verificar o comportamento inicial da simulação distribuída e o segundo para observar a tendência do comportamento com o avançar do tempo de simulação.

6.4.1 Aplicação 1

O comportamento no tempo da simulação distribuída, considerando-se a Aplicação 1, é apresentado nas figuras 6.13 e 6.14, considerando tempo de processamento igual ao padrão e diferentes valores do tempo de comunicação (variando entre 0,25 vezes o padrão até 2 vezes o tempo padrão). Verifica-se uma pequena variação da eficiência, pouco significativa, no decorrer do tempo. Esse comportamento é observado independentemente das características do meio de comunicação utilizado, sendo justificado pelas características da aplicação (uma parte dos clientes deixa o sistema, isto é, nem todos os eventos executados pelo processo lógico 1 escalonam eventos (mensagens) para o processo lógico 2). A diferença na eficiência alcançada entre as plataformas com meios de comunicação distintos será analisada na seção 6.5.

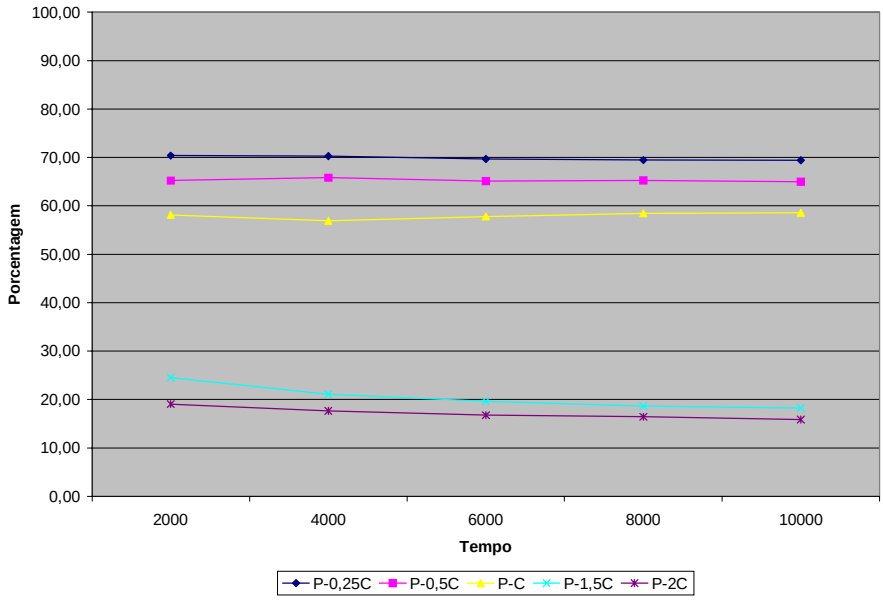


Figura 6.13: Eficiência do Protocolo *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 1 - Intervalo 1).

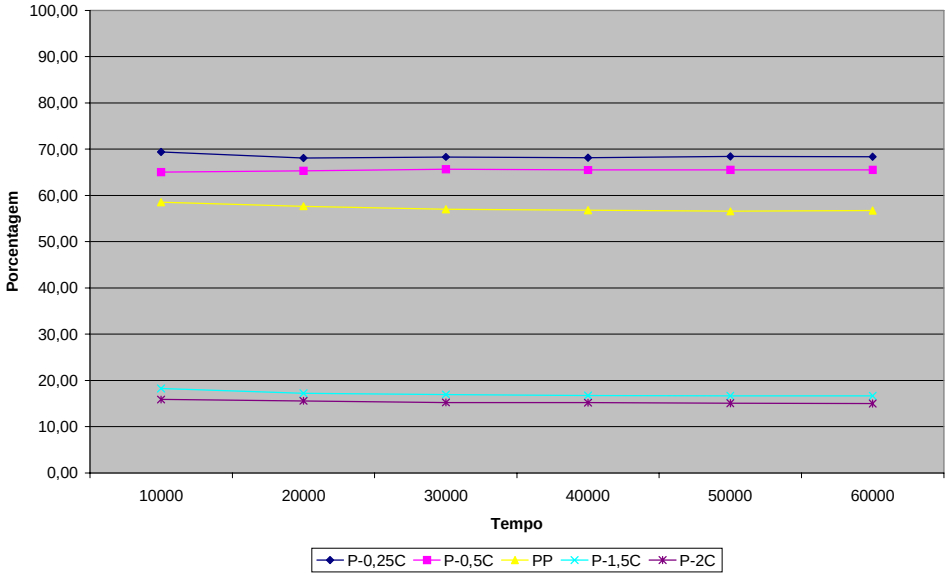


Figura 6.14: Eficiência do Protocolo *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 1-Intervalo 2).

A influência do tempo da simulação distribuída no comprimento médio do *rollback* é apresentada nas figuras 6.15 e 6.16. O comprimento médio do *rollback* sofre variação muito pequena no tempo porque a freqüência (e conseqüentemente o período) dos *rollbacks* praticamente não se altera (figuras 6.17, 6.18, 6.19 e 6.20). O significado do período não se alterar implica que o número médio de eventos

executados entre a ocorrência de um *rollback* e a próxima ocorrência sofre pequena ou nenhuma alteração. Com isso a chance de executar eventos na ordem errada também não muda e o comprimento médio do *rollback* permanece praticamente constante no tempo da simulação distribuída. Essa é uma característica da aplicação, na qual parte dos clientes deixa o sistema (depois de executar no processo lógico 1), implicando em menor número de mensagens enviadas para o processo lógico 2.

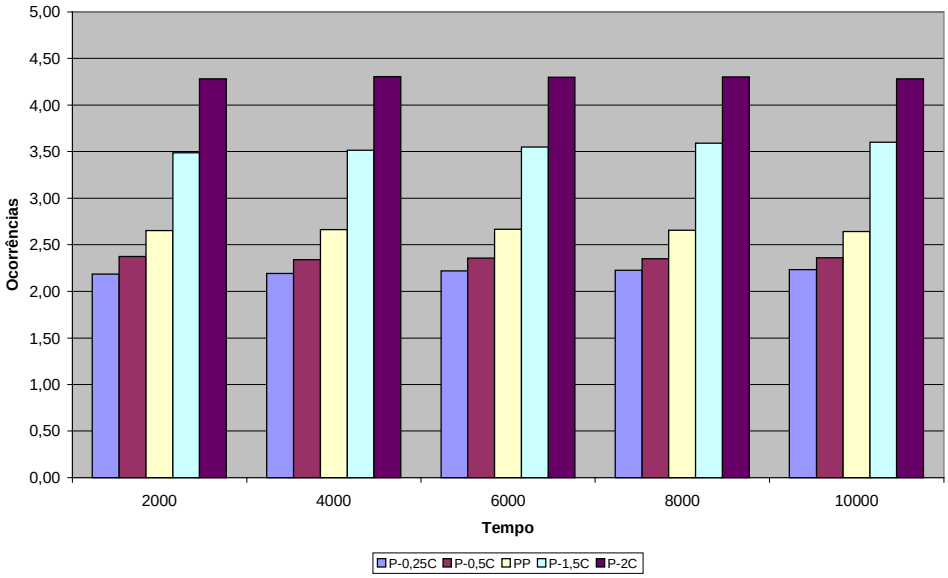


Figura 6.15: Influência do Tempo de Simulação no Comprimento Médio do *Rollback* do *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 1 - Intervalo 1).

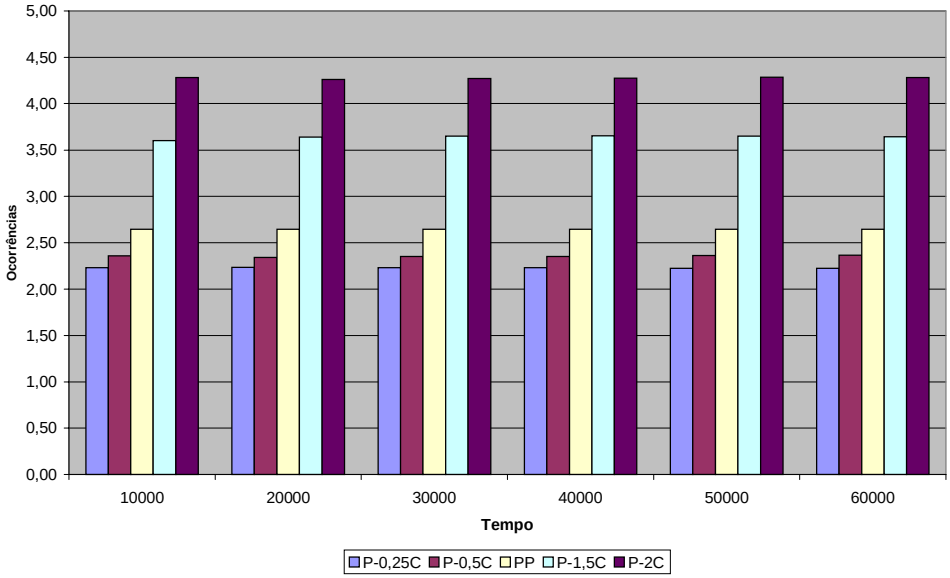


Figura 6.16: Influência do Tempo de Simulação no Comprimento Médio do *Rollback* do *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 1 – Intervalo 2).

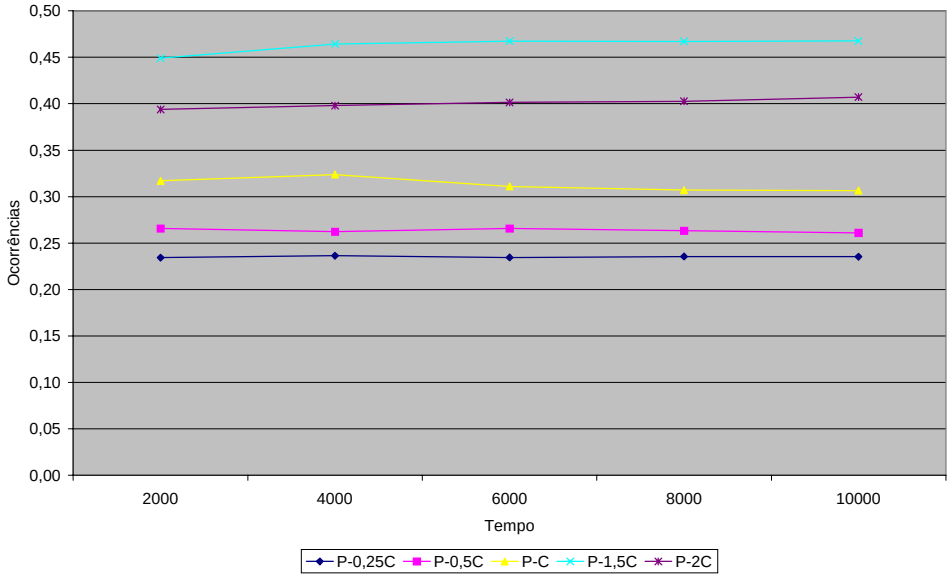


Figura 6.17: Influência do Tempo de Simulação na Freqüência de *Rollbacks* no *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 1 – Intervalo 1).

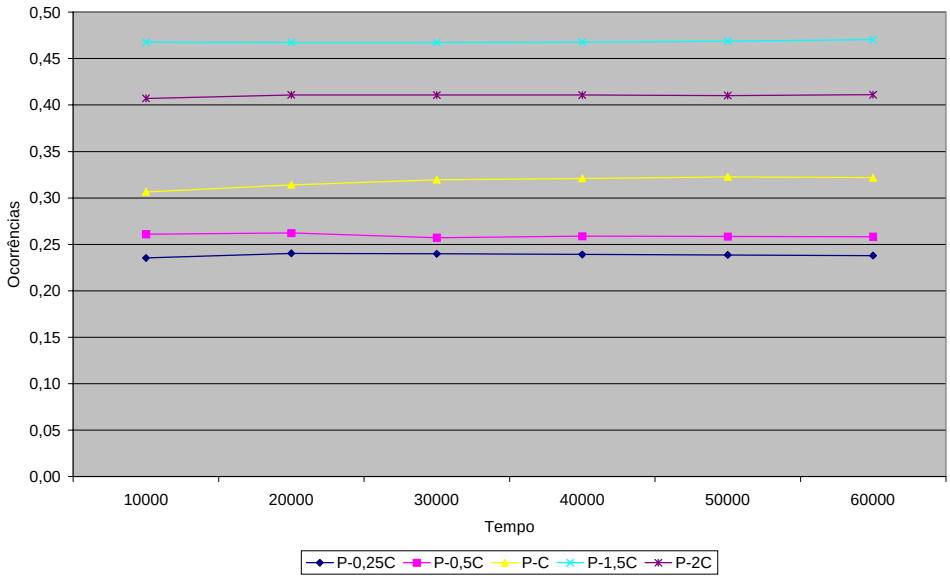


Figura 6.18: Influência do Tempo de Simulação na Freqüência de *Rollbacks* no *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 1 – Intervalo 2).

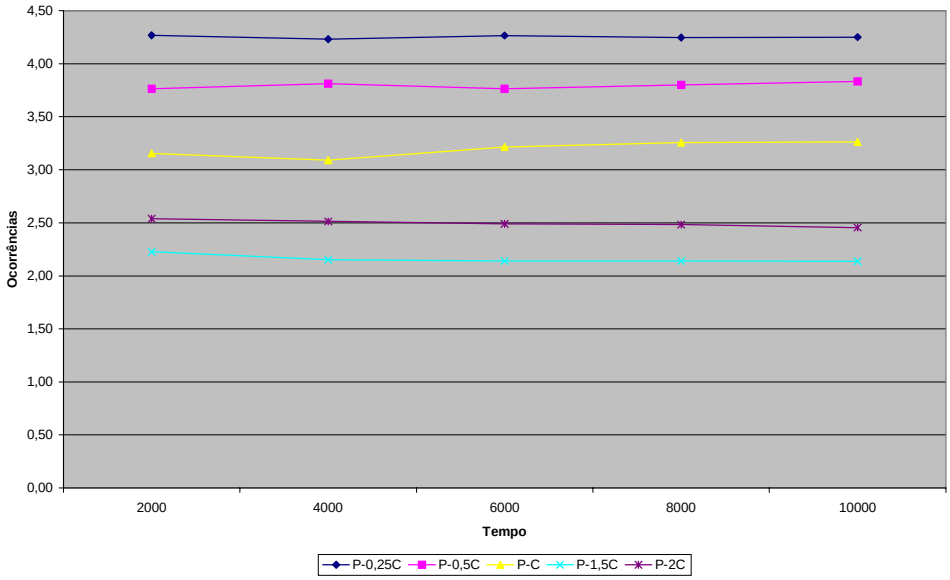


Figura 6.19: Influência do Tempo de Simulação no Período de *Rollbacks* no *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 1 – Intervalo 1).

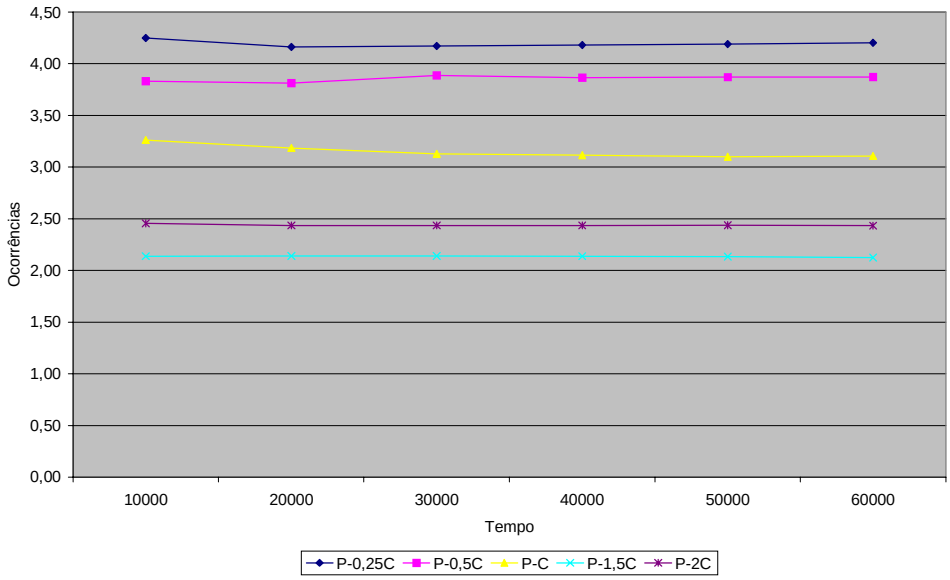


Figura 6.20: Influência do Tempo de Simulação no Período de *Rollbacks* no *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 1 – Intervalo 2).

O comportamento no tempo da simulação distribuída, considerando-se a Aplicação 1, é apresentado nas figuras 6.21 e 6.22, considerando tempo de comunicação igual ao padrão e diferentes valores do tempo de processamento (variando entre 0,25 vezes o padrão até 2 vezes o tempo padrão). Assim como no caso anterior, verifica-se uma pequena variação da eficiência, pouco significativa, no decorrer do tempo. Esse comportamento é observado independentemente das características do meio de processamento utilizado, sendo justificado pelas características da aplicação (uma parte dos clientes deixa o sistema, isto é, nem todos os eventos executados pelo processo lógico 1 escalonam eventos (mensagens) para o processo lógico 2). A diferença na eficiência alcançada entre as plataformas com meios de processamento distintos será analisada na seção 6.5.

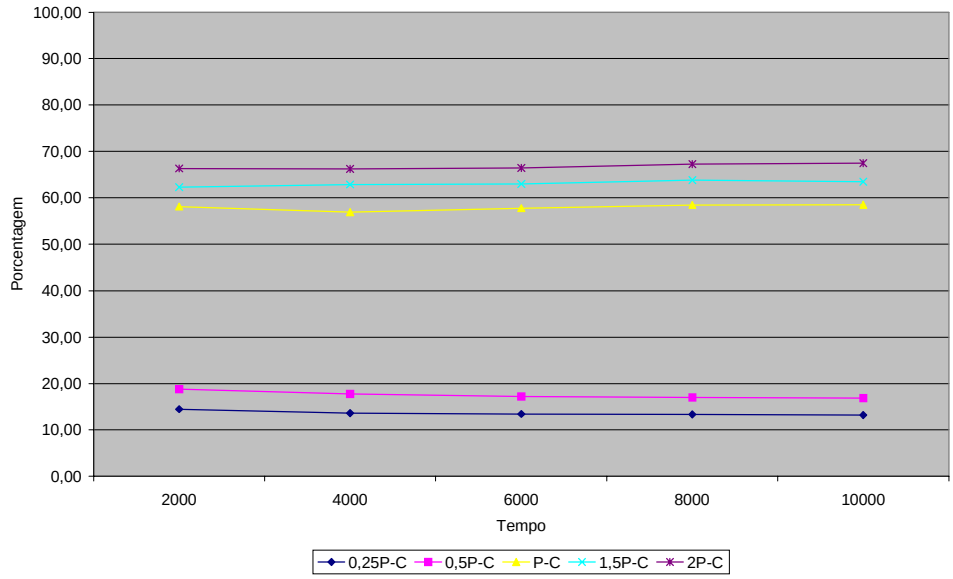


Figura 6.21: Eficiência do Protocolo *Time Warp* Básico em Diferentes Situações do Tempo de Processamento (Aplicação 1 - Intervalo 1).

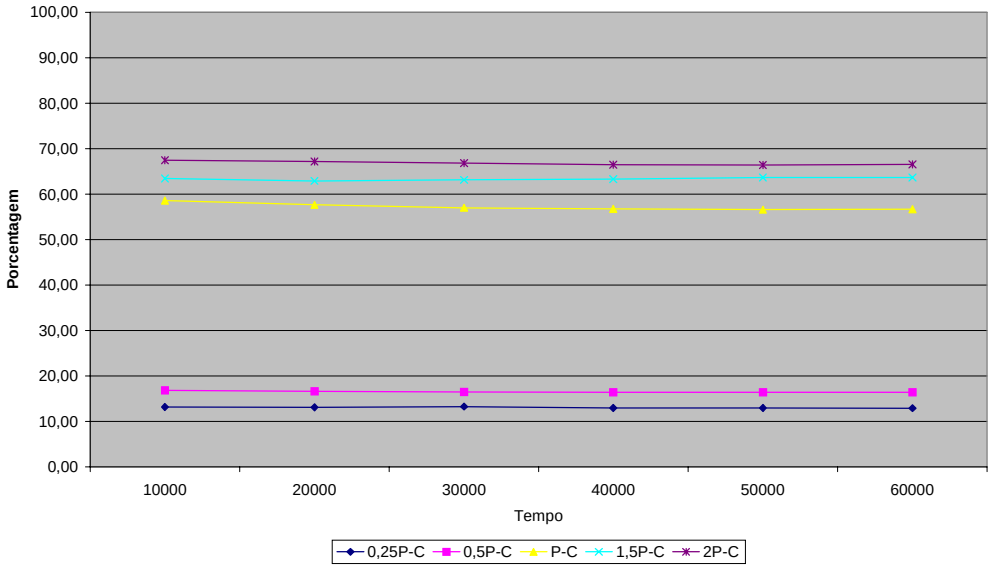


Figura 6.22: Eficiência do Protocolo *Time Warp* Básico em Diferentes Situações do Tempo de Processamento (Aplicação 1 - Intervalo 2).

Da mesma forma que nos casos de teste apresentados anteriormente, o comprimento médio do *rollback* permanece constante no tempo (figuras 6.23 e 6.24), em função da pequena variação da freqüência dos *rollbacks*. Os gráficos da frequência e do período foram omitidos devido ao fato de apresentarem comportamento semelhante aos do tempo com variação no meio de comunicação.

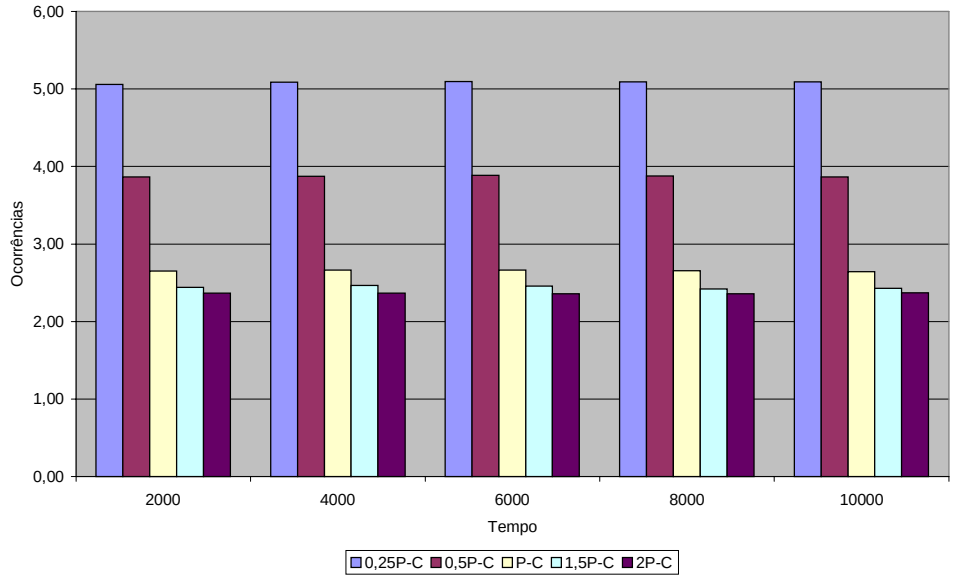


Figura 6.23: Influência do Tempo de Simulação no Comprimento Médio do *Rollback* do *Time Warp* Básico em Diferentes Situações do Tempo de Processamento (Aplicação 1 – Intervalo 1).

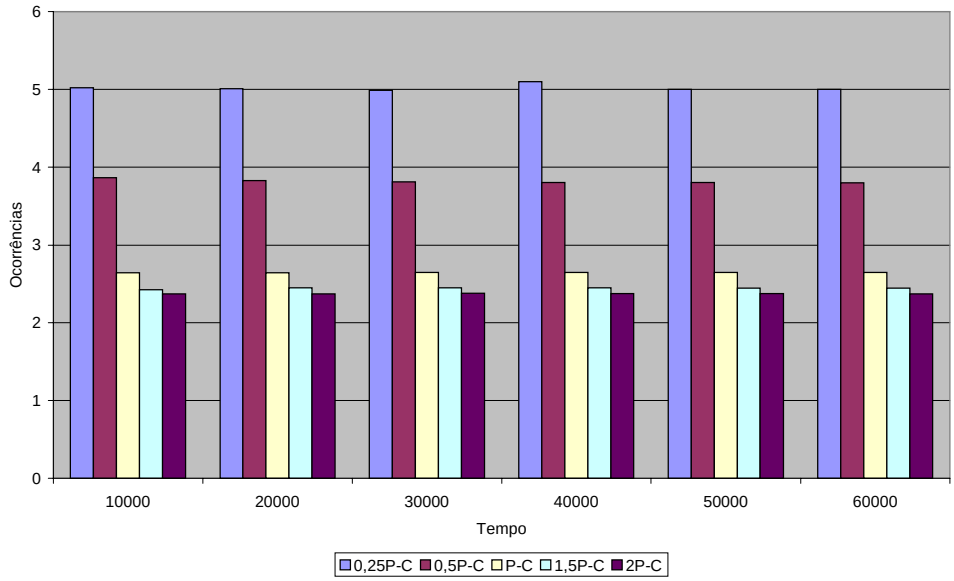


Figura 6.24: Influência do Tempo de Simulação no Comprimento Médio do *Rollback* do *Time Warp* Básico em Diferentes Situações do Tempo de Processamento (Aplicação 1 – Intervalo 2).

6.4.2 Aplicação 2

A influência do tempo na eficiência obtida a partir da simulação distribuída, considerando tempo de processamento igual ao padrão e diferentes valores do tempo de comunicação (variando entre 0,25 vezes o padrão até 2 vezes o tempo padrão), é apresentada nas figuras 6.25 e 6.26. A queda da eficiência, no tempo, ocorre independentemente das características do meio de comunicação e é característica da aplicação, que representa um modelo fechado (todo evento executado em qualquer um dos processos lógicos resulta em eventos escalonados em outro processo lógico). A diferença na eficiência alcançada entre as plataformas com meios de comunicação distintos será analisada na seção 6.5.

A taxa de queda da eficiência diminui com o passar do tempo da simulação distribuída. Por exemplo, com tempo de comunicação igual a 0,25 vezes o tempo padrão, a queda da eficiência do tempo 2000 até 10000 é da ordem de 28%, e de 10000 até 30000 a queda é da ordem de 20%. Nos outros casos a queda é menos acentuada e essa análise será feita na seção 6.5.

Assim como na Aplicação 1, o comprimento médio do *rollback* se mantém constante durante o tempo da simulação distribuída (figuras 6.27 e 6.28). Isso acontece porque, apesar da freqüência dos *rollbacks* aumentar (figuras 6.29 e 6.30) aumentam também os *rollbacks* nos processos lógicos 2 e 3 (cada evento executado no processo lógico 1 gera um outro evento (uma mensagem) que deverá ser executado no processo lógico 2 ou no processo lógico 3). Porém, os *rollbacks* no processo lógico 1 apresentam comprimento médio menor por causa do menor período entre os *rollbacks* (menos eventos executados, menor chance de executar fora de ordem, menos eventos para desfazer). Os *rollbacks* nos processos lógicos 2 e 3 apresentam sempre comprimento igual a 1 (devido à ordem de envio das antimensagens pelo processo lógico 1) e o aumento dos *rollbacks* faz com que esse número influencie mais na média total. Essa é a razão pela qual o comprimento médio se mantém constante também nessa aplicação.

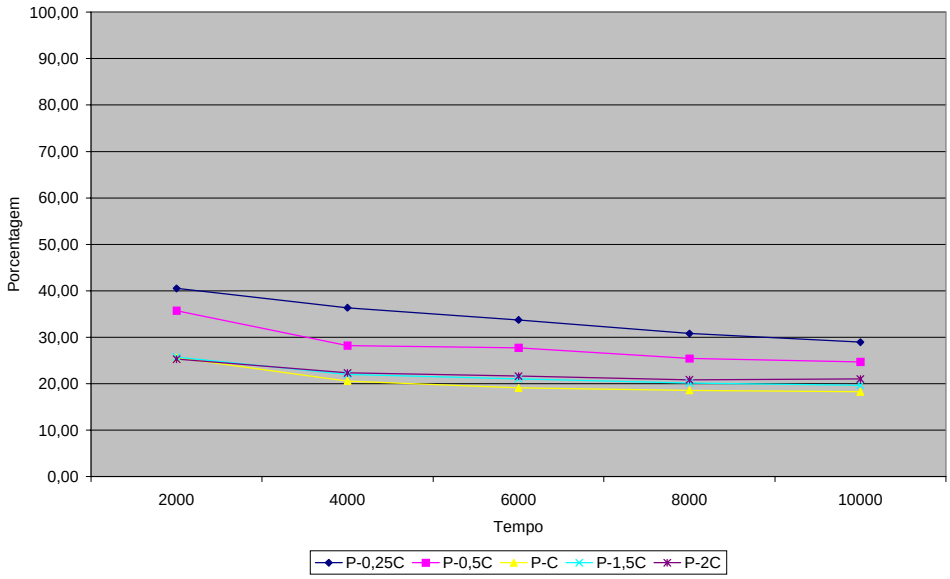


Figura 6.25: Eficiência do Protocolo *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 2 - Intervalo 1).

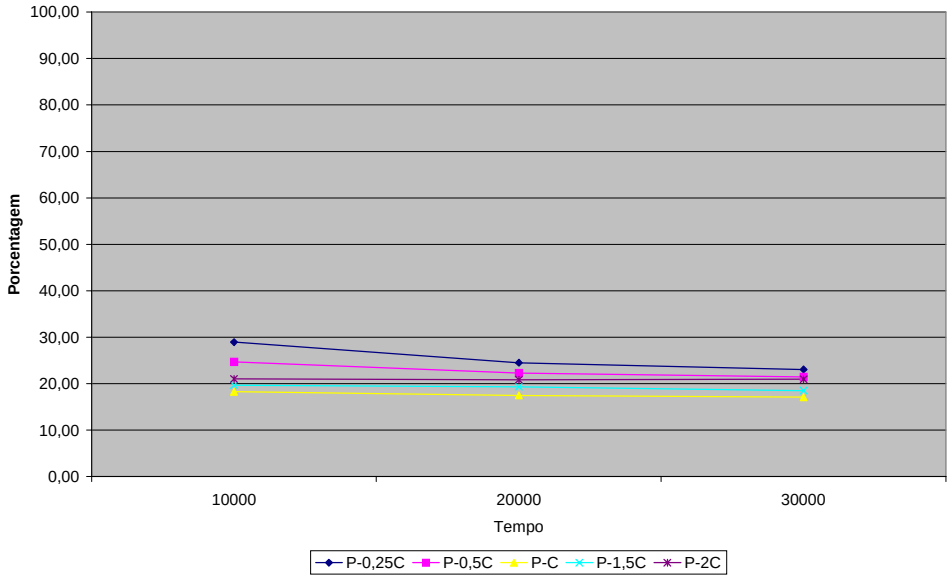


Figura 6.26: Eficiência do Protocolo *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 2 - Intervalo 2).

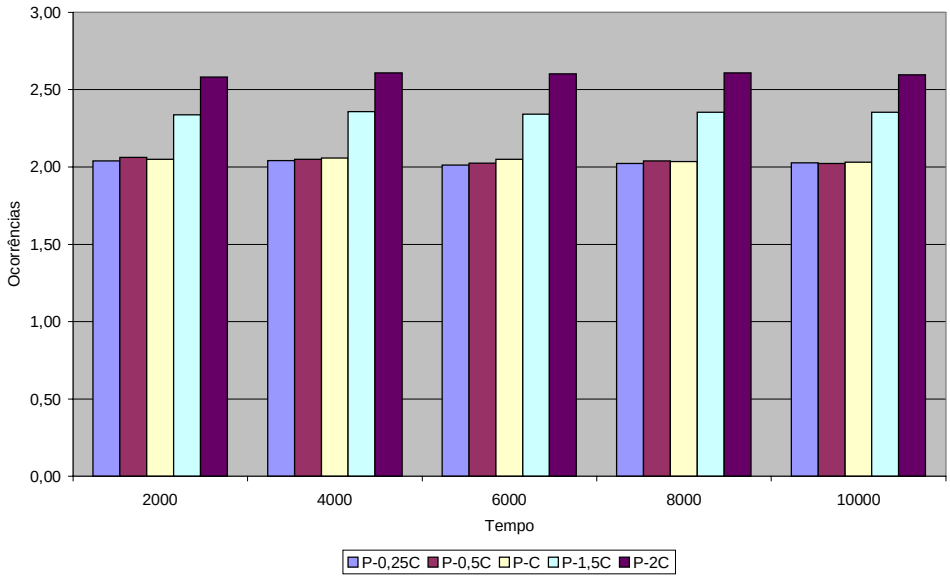


Figura 6.27: Influência do Tempo de Simulação no Comprimento Médio do *Rollback* no *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 2– Intervalo 1).

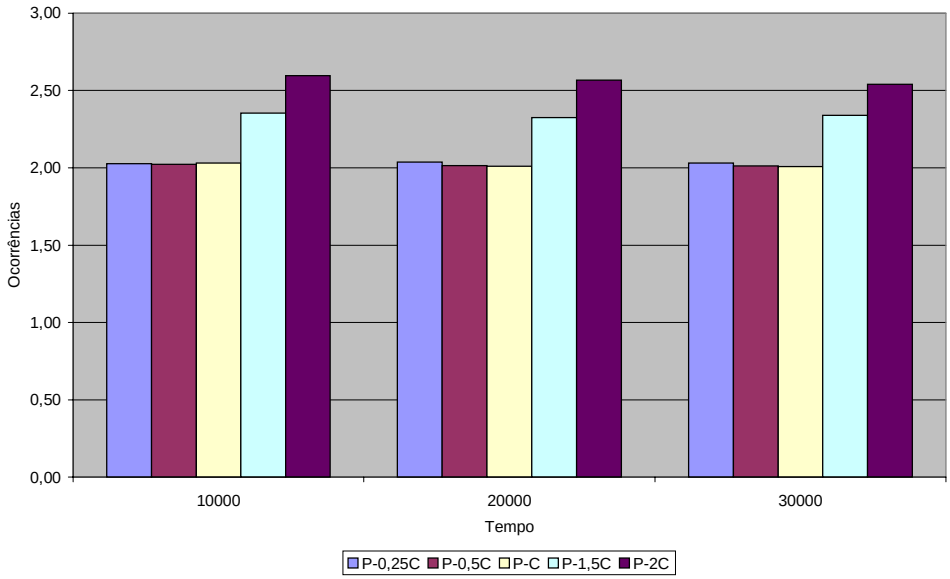


Figura 6.28: Influência do Tempo de Simulação no Comprimento Médio do *Rollback* no *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 2– Intervalo 2).

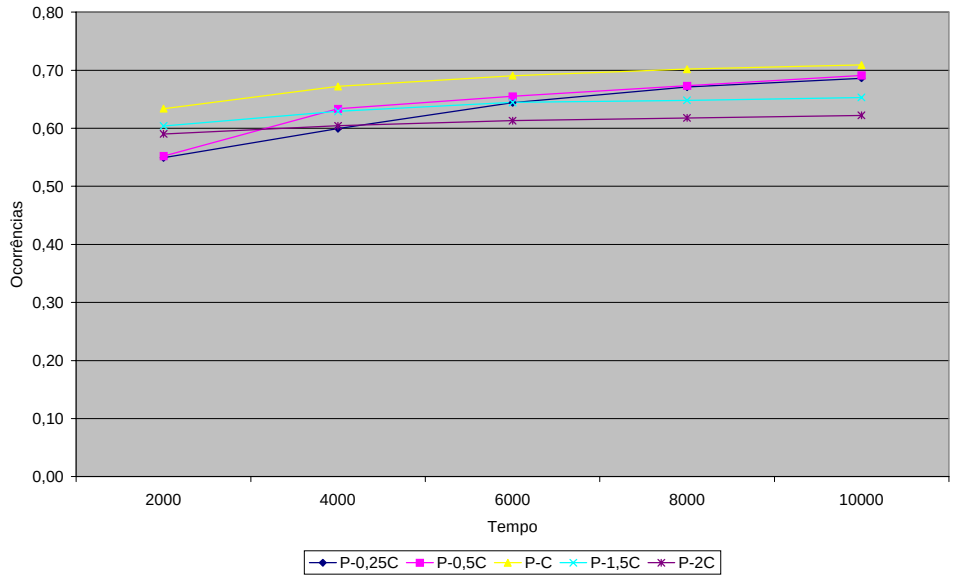


Figura 6.29: Influência do Tempo de Simulação na Freqüencia de *Rollbacks* no *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 2– Intervalo 1).

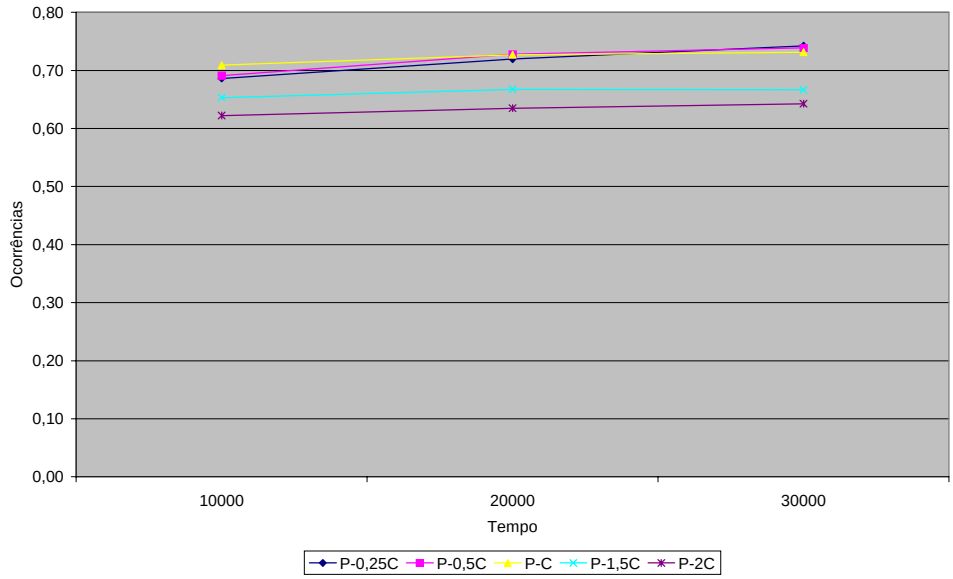


Figura 6.30: Influência do Tempo de Simulação na Freqüência de *Rollbacks* no *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 2– Intervalo 2).

O comportamento da eficiência da simulação distribuída, durante o tempo, considerando-se tempo de comunicação igual ao padrão e diferentes valores do tempo de processamento (variando entre 0,25 vezes o padrão até 2 vezes o tempo

padrão). Independentemente do tempo de processamento utilizado, a tendência da eficiência é diminuir, com taxa de queda decrescente com o avançar do tempo da simulação (figuras 6.31 e 6.32). Essa tendência, da mesma forma que nas situações de variação do parâmetro tempo de comunicação, ocorre em função da característica da aplicação, que representa um sistema fechado. A freqüência dos *rollbacks* aumenta (figuras 6.33 e 6.34) porém o comprimento médio dos *rollbacks* permanece constante, pelo mesmo motivo discutido anteriormente.

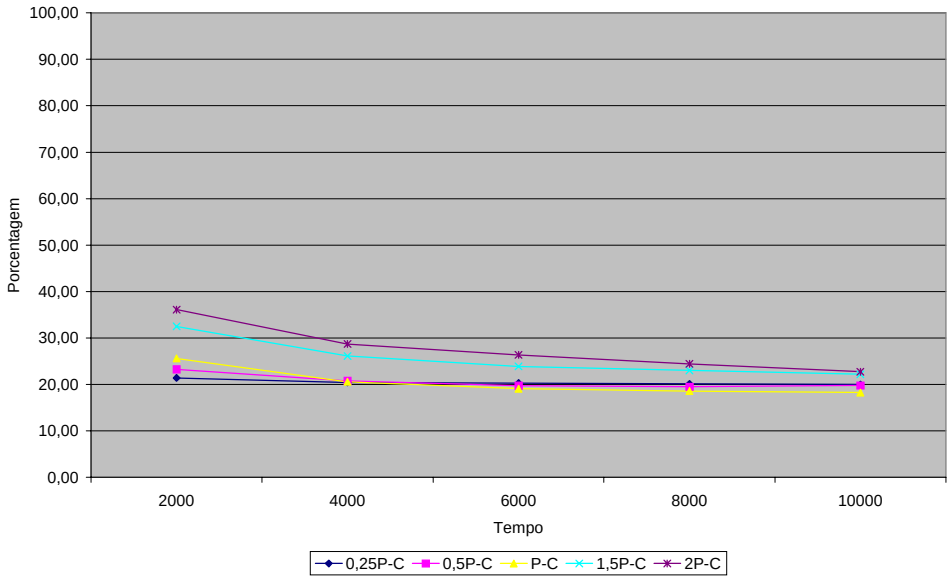


Figura 6.31: Eficiência do Protocolo *Time Warp* Básico em Diferentes Situações do Tempo de Processamento (Aplicação 2-Intervalo 1).

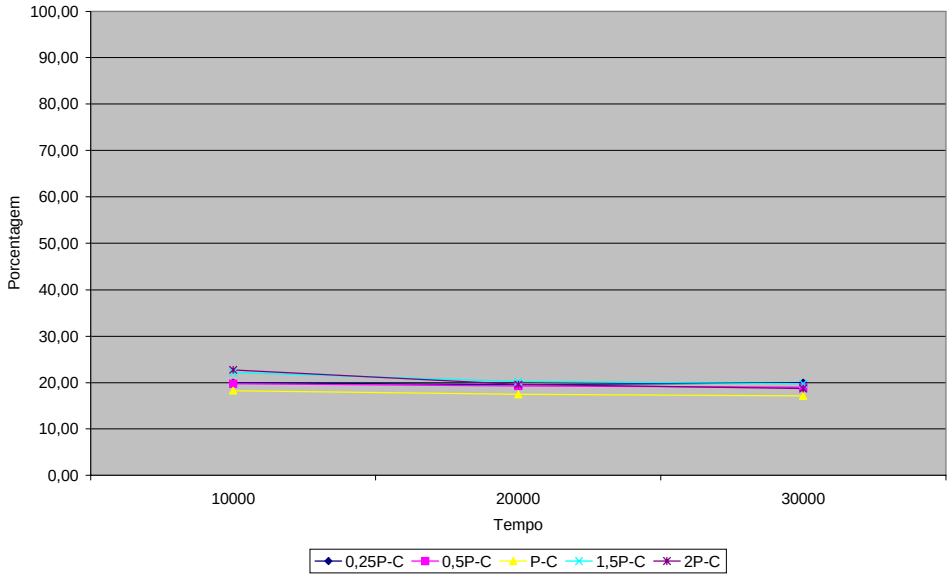


Figura 6.32: Eficiência do Protocolo *Time Warp* Básico em Diferentes Situações do Tempo de Processamento (Aplicação 2 - Intervalo 2).

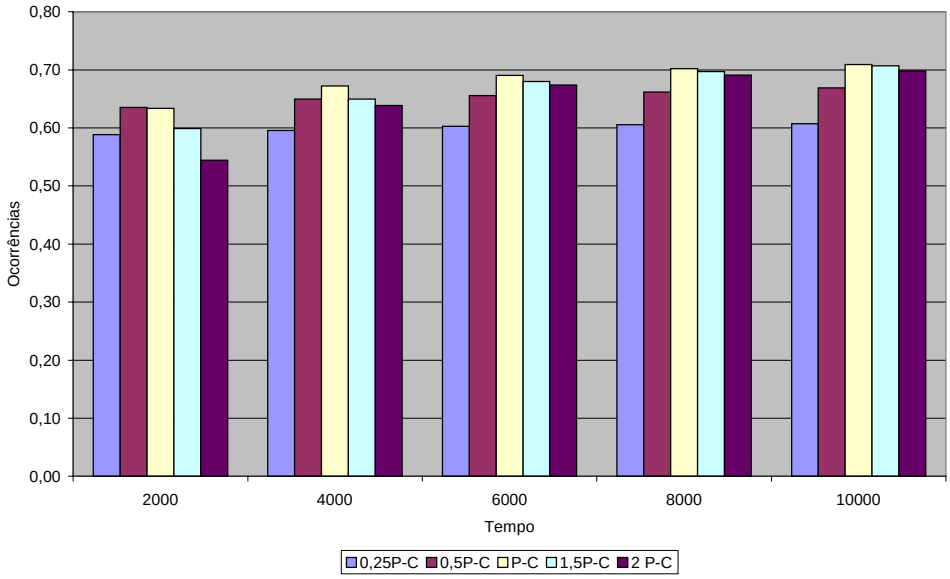


Figura 6.33: Influência do Tempo de Simulação na Freqüência de *Rollbacks* no *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 2 – Intervalo 1).

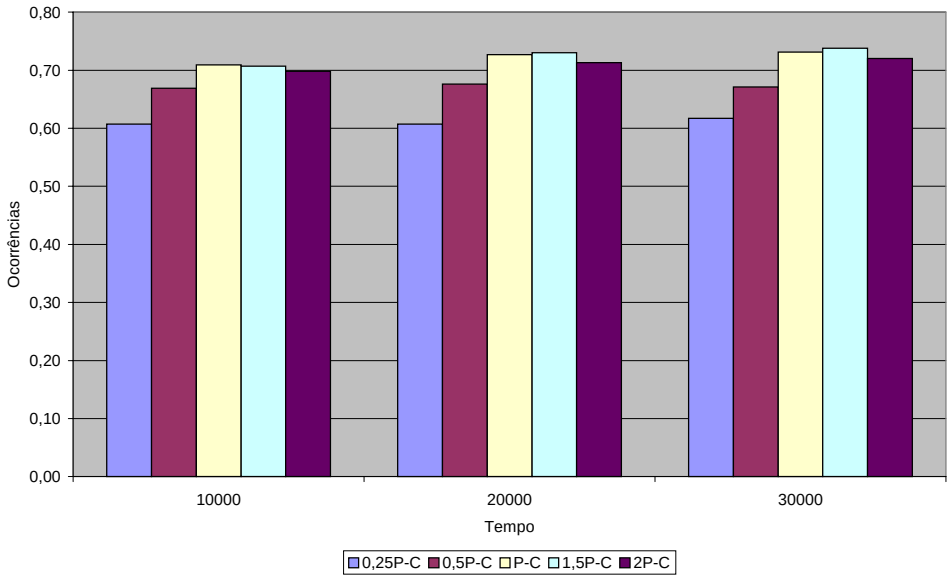


Figura 6.34: Influência do Tempo de Simulação na Freqüência de *Rollbacks* no *Time Warp* Básico em Diferentes Situações do Tempo de Comunicação (Aplicação 2 – Intervalo 2).

6.4.3 Resumo do Estudo do Comportamento do Protocolo no Decorrer do Tempo da Simulação Distribuída

A análise da evolução, no tempo, do comportamento da simulação distribuída permitiu verificar que:

- A eficiência é praticamente estável na Aplicação 1 e apresenta queda na Aplicação 2. Essa diferença é resultado do tipo de sistema que as aplicações representam (misto e fechado, respectivamente), que faz com que a freqüência dos *rollbacks* permaneça praticamente constante na Aplicação 1 (devido ao fato de que parte dos eventos executados no processo lógico 1 não gera eventos para o processo lógico 2, o que significa menor chance de provocoar *rollback* em ambos processos lógicos) e tenda a aumentar na Aplicação 2 (cada evento no processo lógico 1 gera um evento para um dos outros dois processos lógicos);
- Utilizando-se a métrica eficiência, pode-se ter uma idéia de quando utilizar o mecanismo de troca de protocolo proposto por Morselli (Morselli 2000, Morselli et al. 2000). Na Aplicação 1 a qualquer

momento do tempo pode-se efetuar a troca, sendo necessário apenas decidir o que é uma boa eficiência (foi mostrado que as características da plataforma influenciam na eficiência, apesar de permanecer constante no tempo). Na Aplicação 2 a troca pode ser efetuada quando a eficiência atingir um limite crítico, que também deve ser definido;

- O comprimento médio do *rollback* é estável no tempo, independentemente do tipo de sistema que a aplicação representa.

6.5 Estudo da Influência da Arquitetura na Simulação Distribuída

Nesta seção é analisado o impacto da alteração nos tempos de processamento e comunicação na simulação distribuída, considerando as Aplicações 1 e 2.

6.5.1 Influência da Variação do Tempo de Comunicação

6.5.1.1 Aplicação 1

Para verificar a influência da variação do tempo de comunicação na simulação distribuída foram executados experimentos onde esse tempo variou de 0,1 vezes até 2 vezes o tempo de comunicação padrão discutido na seção 6.3.1. Repetiu-se o experimento para diferentes situações do tempo de processamento (tempo de processamento reduzido à metade – série 0,5P do gráfico da figura 6.35, tempo de processamento padrão – série P, tempo de processamento uma vez e meia maior em relação ao padrão – série 1,5P, e tempo de processamento duas vezes maior do que o padrão – série 2P).

Observa-se que, independentemente do tempo de processamento utilizado, quando o meio de comunicação é muito rápido a eficiência máxima obtida é de 70%. Porém, conforme o tempo de comunicação aumenta, a tendência da eficiência é diminuir.

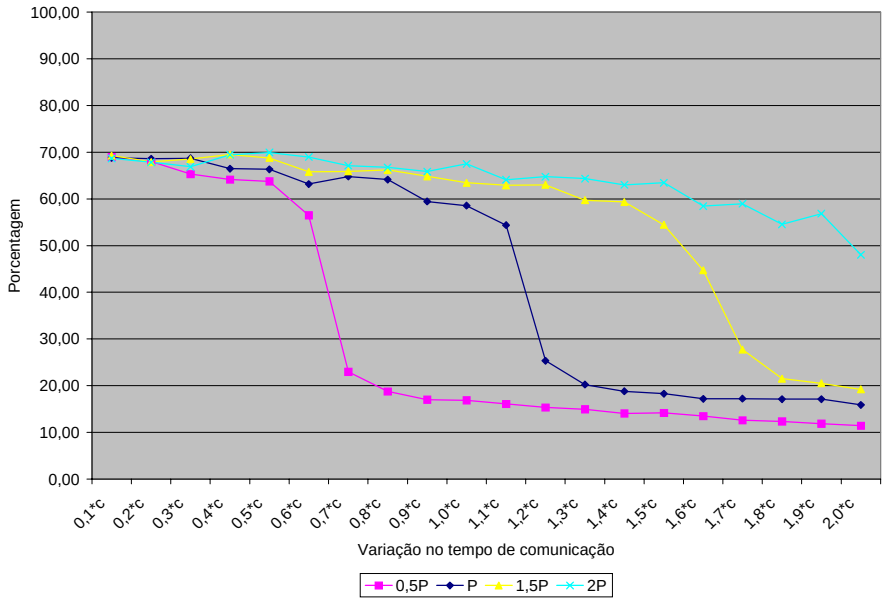


Figura 6.35: Influência da Variação do Tempo de Comunicação na Eficiência do *Time Warp* Básico (Aplicação 1).

A queda brusca da eficiência verificada na figura 6.35 ocorre devido ao aumento substancial do número de *rollbacks*, como mostrado na figura 6.36. Esse aumento acontece devido à característica da aplicação que representa um sistema aberto. Essa situação é explicada devido ao fato de que nesse caso existe um equilíbrio entre a geração de novas mensagens e as mensagens que são enviadas pelo processo lógico 1 para o processo lógico 2.

Com o meio de comunicação mais lento, ocorre um desequilíbrio e a simulação distribuída executa (no processo lógico 1) primeiro mais eventos relativos à chegada de novos clientes do que eventos oriundos do processo lógico 2. Nesse caso aumenta a freqüência de *rollbacks* e a eficiência se reduz. Existe uma combinação dos valores dos tempos de comunicação e de processamento de forma que as mensagens que são inseridas no *Buffer* de Entrada do processo lógico 1, pelo *plugin Gerador de Chegadas*, interferem de forma mais significativa na execução do processo lógico 1 (devido ao fato das marcas de tempo das mensagens de chegada e das mensagens oriundas do processo lógico 1 para o processo lógico 2 terem construções distintas). Os eventos relativos às novas chegadas acontecem sempre em função do *LVT* do processo lógico 1 (que é atualizado a cada execução de evento) enquanto os demais acontecem em função dos parâmetros de utilização dos recursos da aplicação. Com o aumento do tempo de comunicação esses efeitos vão se

tornando maiores e o número de *rollbacks* vai aumentando até atingir um máximo. A partir daí, o tempo de comunicação maior faz com que, no mesmo período de observação, o número de eventos executados diminua e, com isso, o número de *rollbacks*. O número de eventos executados corretamente passa a diminuir porque o comprimento médio do *rollback* aumenta (novos clientes da aplicação cujos eventos são executados enquanto as mensagens vindas do Processo Lógico 2 demoram a chegar) (figuras 6.37 e 6.38).

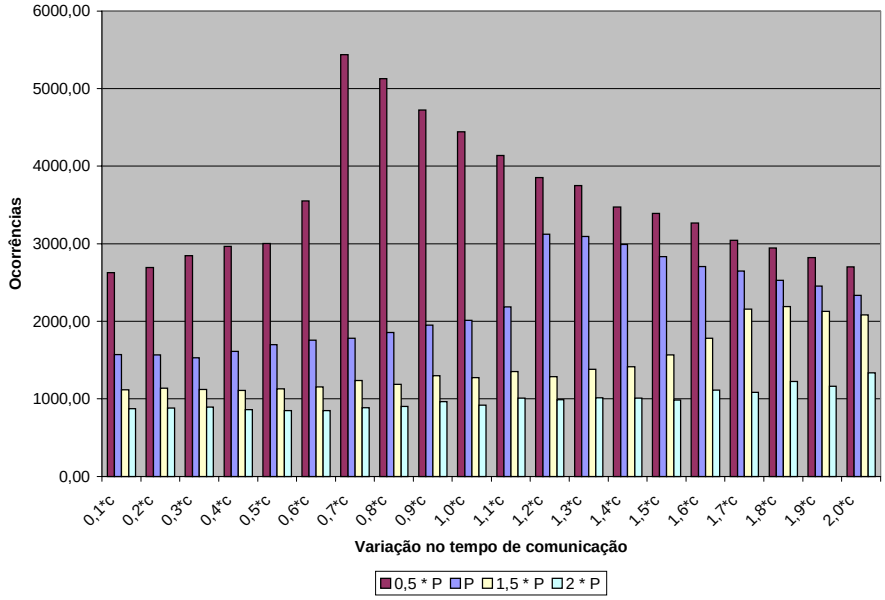


Figura 6.36: Influência da Variação do Tempo de Comunicação no Número de *Rollbacks* Ocorridos no *Time Warp* Básico (Aplicação 1).

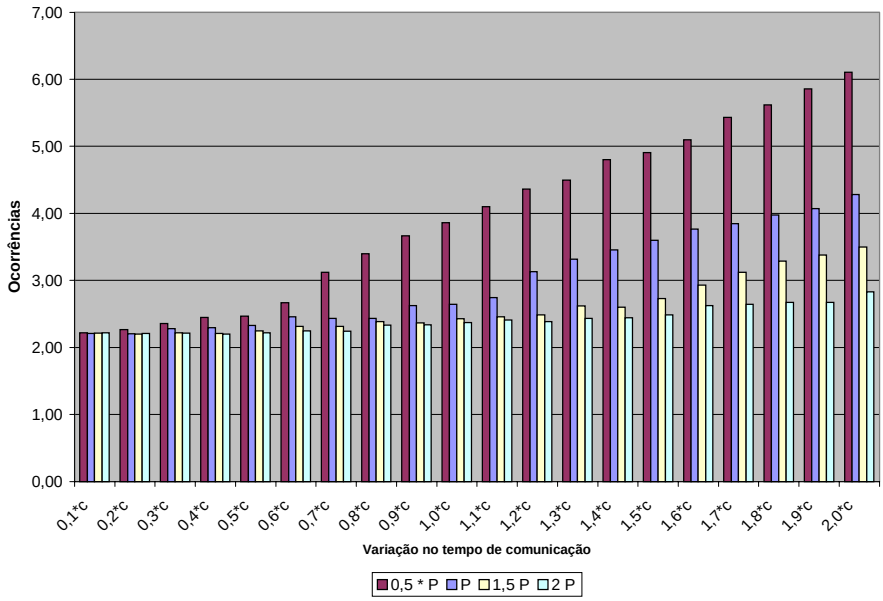


Figura 6.37: Influência da Variação do Tempo de Comunicação no Comprimento Médio do *Rollback* no *Time Warp* Básico (Aplicação 1).

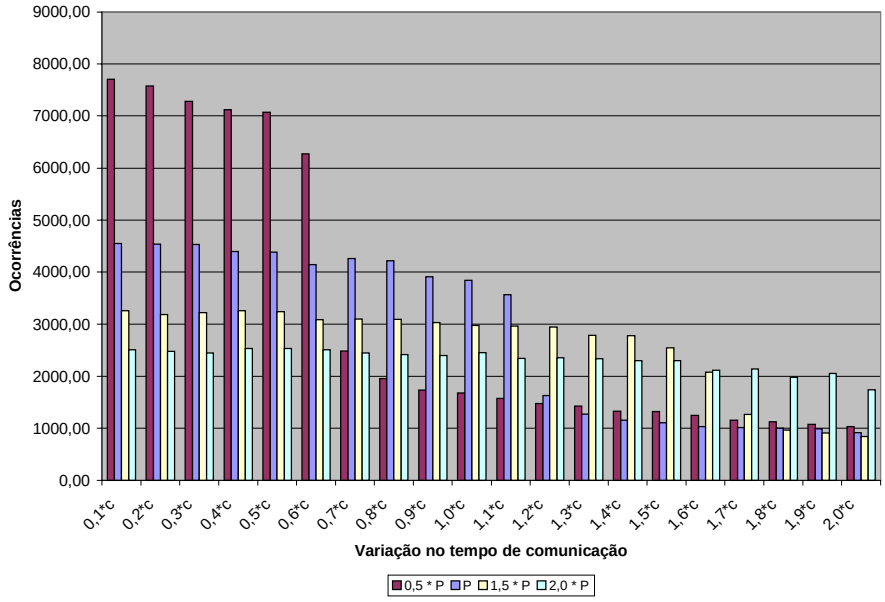
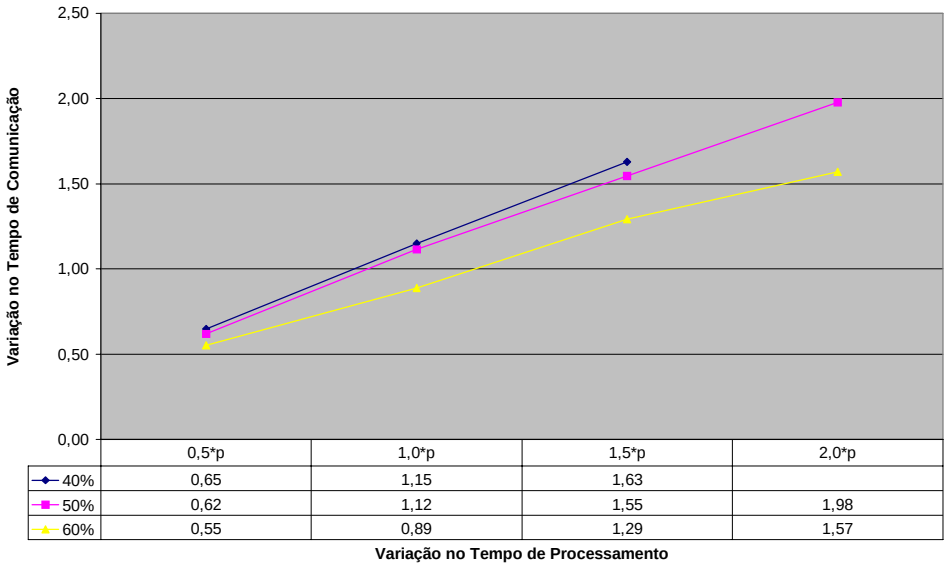


Figura 6.38: Influência da Variação do Tempo de Comunicação no Número de Eventos Executados Corretamente no *Time Warp* Básico (Aplicação 1).

A partir dos resultados de desempenho apresentados, pode-se construir um guia de referência para que o usuário da simulação distribuída possa ter noção da eficiência que a solução da sua aplicação, usando o *Time Warp* básico, pode atingir, dependendo da configuração da plataforma disponível. Esse guia está apresentado

na figura 6.39, que mostra os valores dos tempos de comunicação e processamento necessários para atingir eficiências de 40%, 50% e 60%. Quando se tem disponível uma plataforma onde o tempo de processamento é duas vezes superior ao tempo padrão, não é possível atingir eficiência de 40% e portanto essa situação não está representada na figura 6.39.



	0,5*p	1,0*p	1,5*p	2,0*p
40%	0,65	1,15	1,63	
50%	0,62	1,12	1,55	1,98
60%	0,55	0,89	1,29	1,57

Figura 6.39: Diretrizes para se Obter Eficiência entre 40% e 60% Utilizando *Time Warp* (Aplicação 1).

6.5.1.2 Aplicação 2

Para verificar a influência da variação do tempo de comunicação na simulação distribuída foram executados experimentos onde esse tempo variou de 0,2 vezes até 2 vezes o tempo de comunicação padrão discutido na seção 6.3.1. Repetiu-se o experimento para diferentes situações do tempo de processamento (as séries do gráfico da figura 6.40 apresentam tempo de processamento reduzido a um quarto – série 0,25P, tempo de processamento reduzido à metade – série 0,5P, tempo de processamento padrão – série P, tempo de processamento uma vez e meia maior em relação ao padrão – série 1,5P, e tempo de processamento duas vezes maior do que o padrão – série 2P).

Observa-se que, independentemente do tempo de processamento utilizado, quando o meio de comunicação é muito rápido a eficiência máxima obtida está em

torno de 25%. Porém, conforme o tempo de comunicação aumenta, a tendência da eficiência é diminuir e atingir valores próximos a 20%. Com o aumento do tempo de comunicação o número de eventos executados tende a diminuir, porque o período de observação da simulação foi mantido igual em todas as observações. Dessa forma a ocorrência de *rollbacks* tende a diminuir, porém, o seu comprimento médio aumenta porque os processos lógicos demoram a receber as antimensagens para desfazer computações errôneas (figuras 6.41 e 6.42). Com o aumento do comprimento médio do *rollback* a ocorrência de eventos executados corretamente tende a diminuir porque o processo lógico tem mais erros para corrigir (figura 6.43).

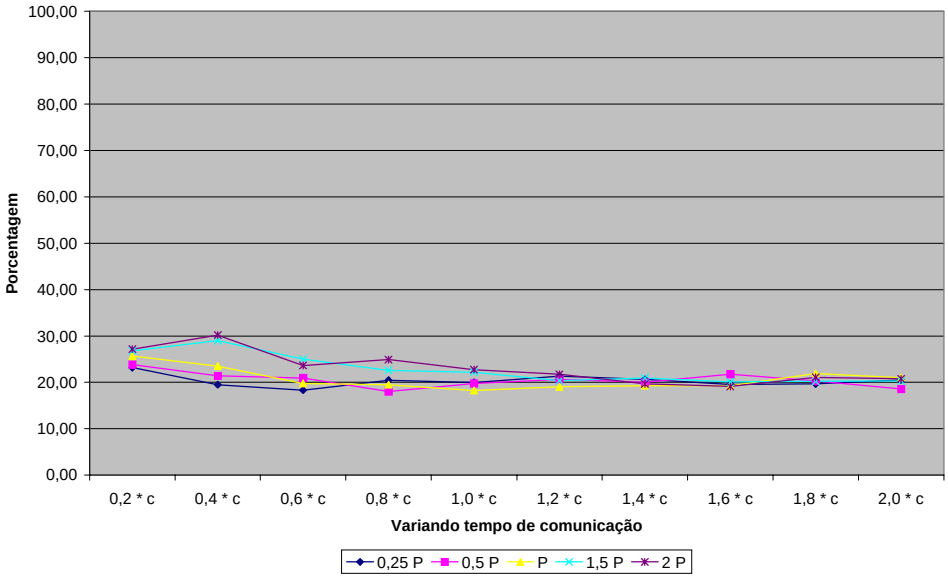


Figura 6.40: Influência da Variação do Tempo de Comunicação na Eficiência do *Time Warp* Básico (Aplicação 2).

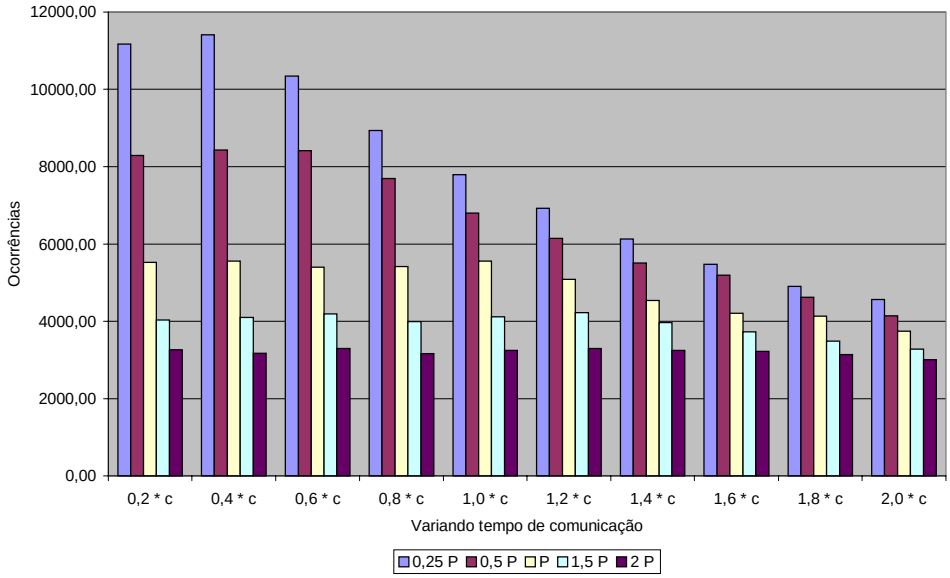


Figura 6.41: Influência da Variação do Tempo de Comunicação no Número de *Rollbacks* do *Time Warp* Básico (Aplicação 2).

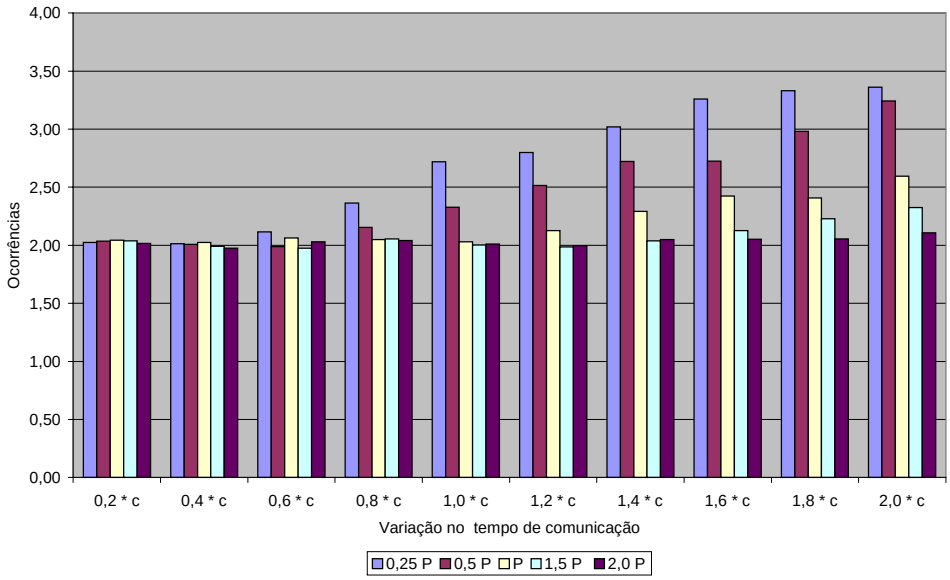


Figura 6.42: Influência da Variação do Tempo de Comunicação no Comprimento Médio do *Rollback* no *Time Warp* Básico (Aplicação 2).

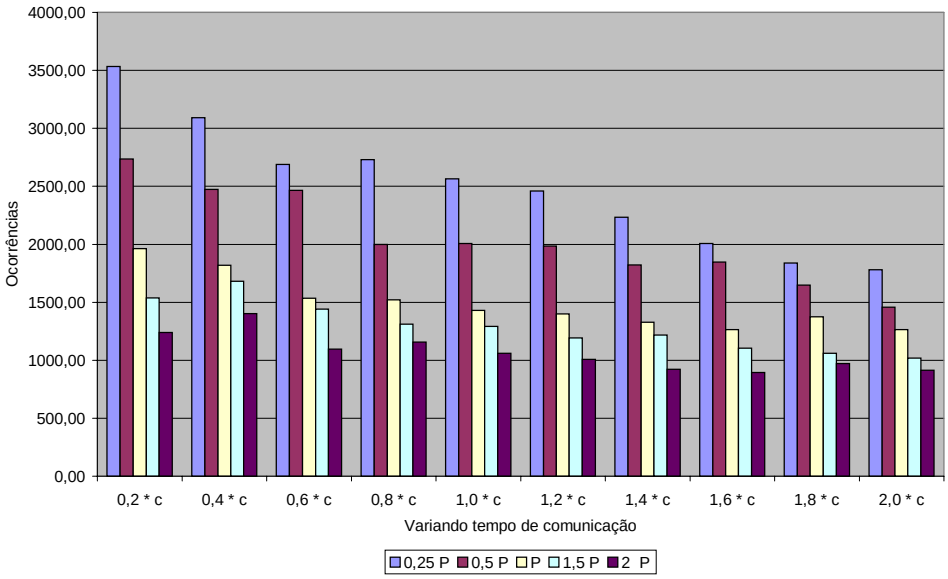


Figura 6.43: Influência da Variação do Tempo de Comunicação no Número de Eventos Executados Corretamente no *Time Warp* Básico (Aplicação 2).

6.5.2 Influência da Variação do Tempo de Processamento

6.5.2.1 Aplicação 1

Para verificar a influência da variação do tempo de processamento na simulação distribuída foram executados experimentos onde esse tempo variou de 0,1 vezes até 2 vezes o tempo de processamento padrão discutido na seção 6.3.1. Repetiu-se o experimento para diferentes situações do tempo de comunicação (tempo de comunicação reduzido a metade – série 0,5C, tempo de comunicação padrão – série C, tempo de processamento uma vez e meia maior em relação ao padrão – série 1,5C, e tempo de processamento duas vezes maior do que o padrão – série 2C do gráfico da figura 6.44).

O gráfico da figura 6.44 mostra que a eficiência da simulação distribuída aumenta conforme o tempo de processamento aumenta, independentemente do tempo de comunicação e tende a um valor constante. Os valores do tempo de processamento nos quais a eficiência tende a atingir esse valor constante dependem diretamente do tempo de comunicação (quanto maior o tempo de comunicação maior é o tempo de processamento para se atingir esse valor constante). A eficiência aumenta porque o tempo de processamento limita a execução de eventos e portanto a

probabilidade de executar eventos fora da ordem cronológica. (figura 6.45). O número total de *rollbacks* ocorridos e seu comprimento médio diminui (figura 6.46).

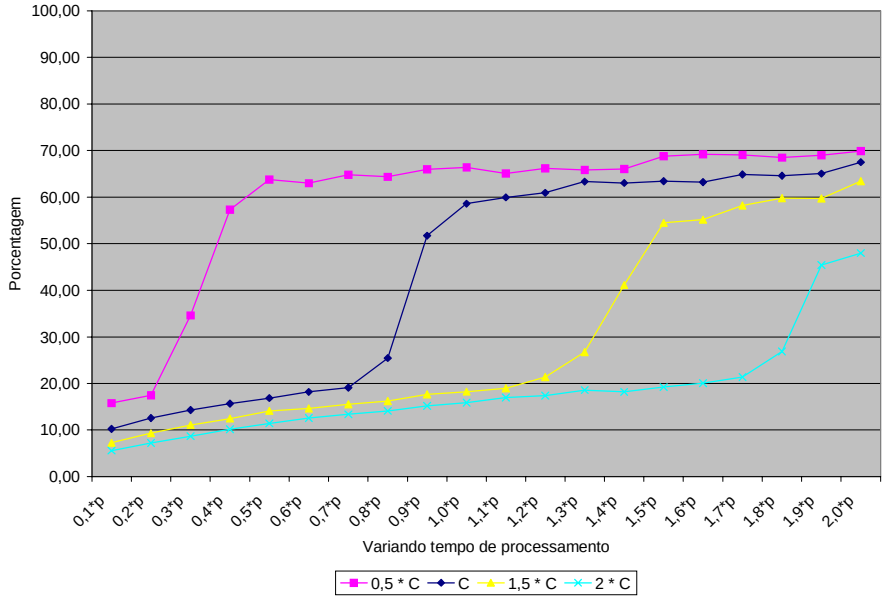


Figura 6.44: Influência da Variação do Tempo de Processamento na Eficiência do *Time Warp* Básico (Aplicação 1).

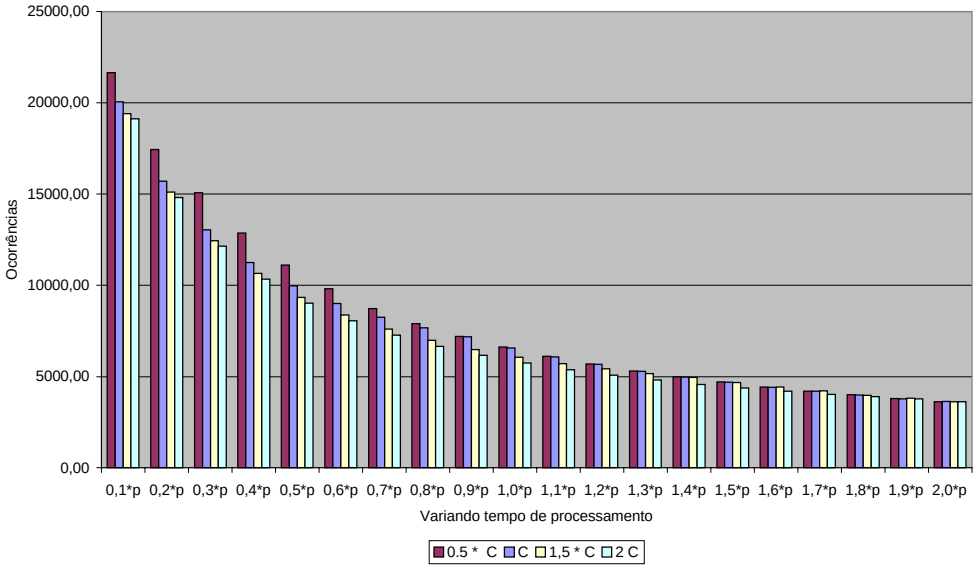


Figura 6.45: Influência da Variação do Tempo de Processamento no Número de Eventos Executados no *Time Warp* Básico (Aplicação 1).

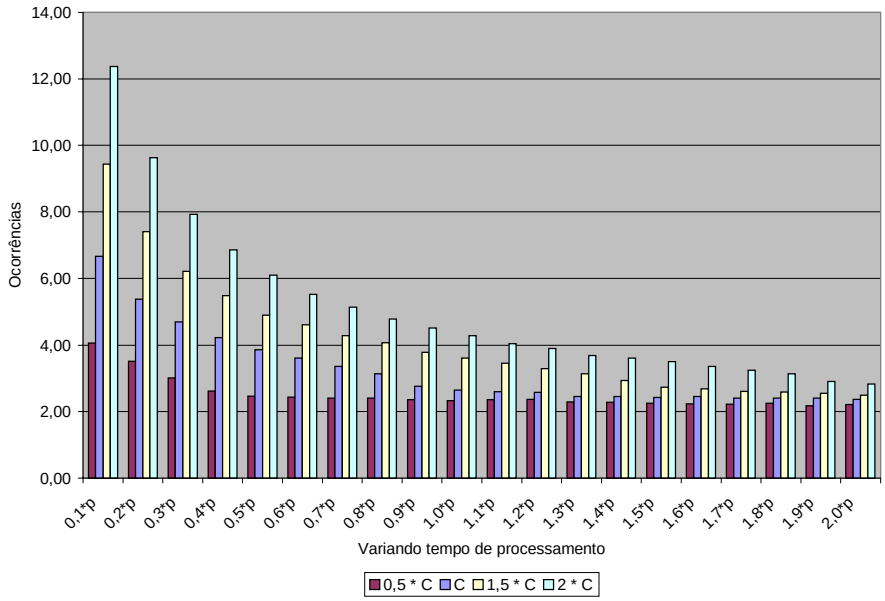


Figura 6.46: Influência da Variação do Tempo de Processamento no Comprimento Médio do *Rollback* do *Time Warp* Básico (Aplicação 1).

A partir dos resultados de desempenho apresentados, foi construído um guia de referência para que o usuário da simulação distribuída possa ter noção da eficiência que a solução da sua aplicação, resolvida através *Time Warp* básico, pode atingir, dependendo da configuração da plataforma disponível. Esse guia foi apresentado na figura 6.39, que mostra os valores dos tempos de comunicação e processamento necessários para atingir eficiências de 40%, 50% e 60%.

6.5.2.2 Aplicação 2

Para verificar a influência da variação do tempo de processamento na simulação distribuída foram executados experimentos onde esse tempo variou de 0,2 vezes até 2 vezes o tempo de processamento padrão discutido na seção 6.3.1. Repetiu-se o experimento para diferentes situações do tempo de comunicação (séries do gráfico da figura 6.47: tempo de comunicação reduzido a um quarto – série 0,25C, tempo de comunicação reduzido a metade – série 0,5C, tempo de comunicação padrão – série C, tempo de comunicação uma vez e meia maior em relação ao padrão – série 1,5C, e tempo de comunicação duas vezes maior do que o padrão – série 2C).

Os resultados mostram que a eficiência permanece entre 20 e 30%, devido ao número fixo de mensagens correspondendo aos eventos de clientes (tarefas) que circulam pela aplicação (entre a UCP e as unidades de Disco). Com o aumento do tempo de processamento, um menor número de eventos é executado, o que implica na diminuição dos eventos executados corretamente e do número de *rollbacks*.

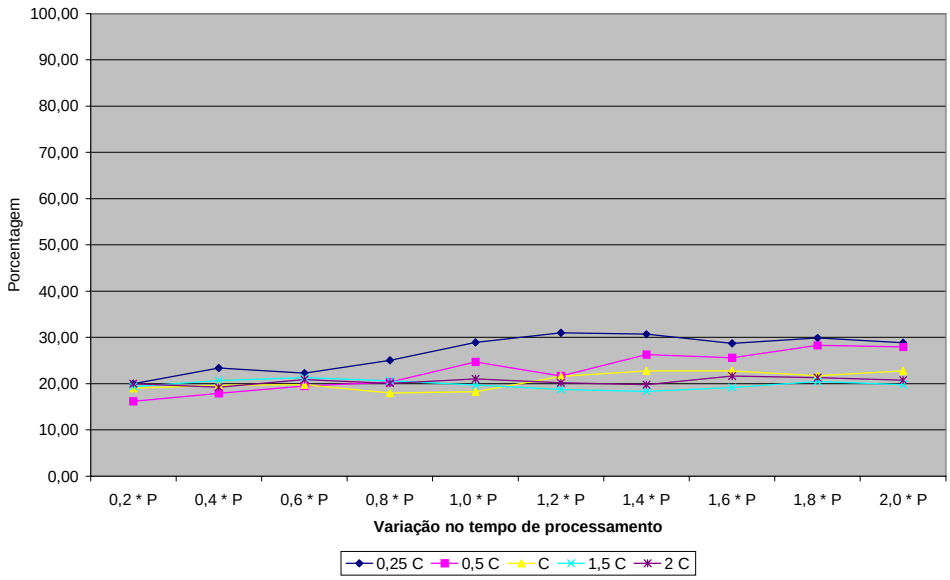


Figura 6.47: Influência da Variação do Tempo de Processamento na Eficiência do *Time Warp* Básico (Aplicação 2).

6.5.3 Resumo do Estudo da Influência dos Parâmetros da Plataforma no Comportamento da Simulação Distribuída

As figuras 6.48 e 6.49 mostram a relação entre o parâmetro tempo de comunicação e as métricas comprimento médio do *rollback* e eficiência nas Aplicações 1 e 2, respectivamente. As características da aplicação 1, representando um sistema misto, fazem com que alterações na característica da plataforma de hardware impliquem em redução da eficiência (da ordem de 70%), devido à maior influência dos eventos representando as chegadas de novos clientes ao sistema. No caso da Aplicação 2, o sistema fechado com número fixo de clientes provoca uma redução máxima da eficiência da ordem de 20%. Para diferenciar as duas situações, na figura 6.48 a eficiência é apresentada em queda enquanto na figura 6.49 é apresentada estacionária para mostrar que a queda é menor.

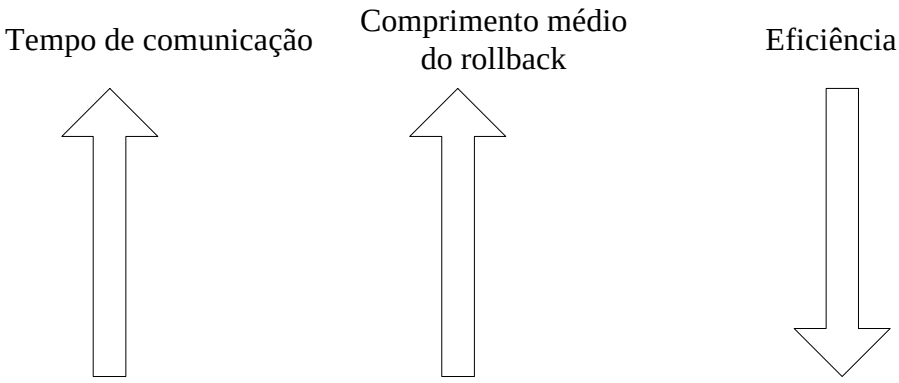


Figura 6.48: Representação da Relação entre Tempo de Comunicação, Eficiência e Comprimento Médio do *Rollback Time Warp* Básico (Aplicação 1).

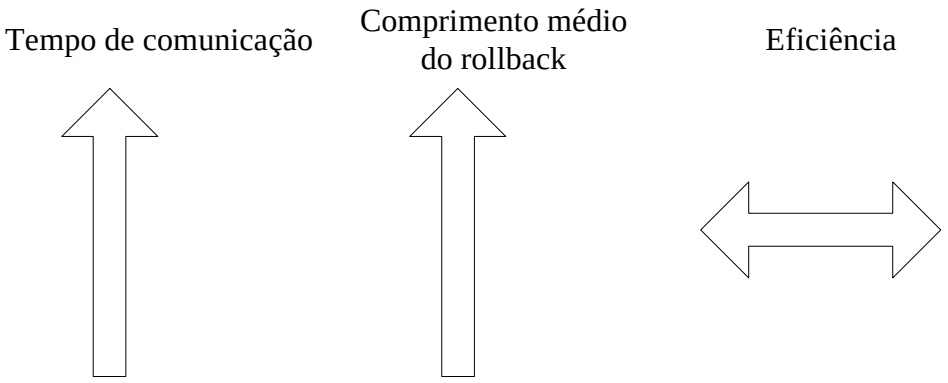


Figura 6.49: Representação da Relação entre Tempo de Comunicação, Eficiência e Comprimento Médio do *Rollback Time Warp* Básico (Aplicação 2).

A variação do tempo de processamento também influencia, de forma significativa, a eficiência na Aplicação 1, que pode aumentar em 350% (figura 6.50). Com o aumento do tempo de processamento, melhora a eficiência da simulação distribuída porque diminui o comprimento médio do *rollback*, em função das características da aplicação, que representa um sistema misto. No caso da Aplicação 2, da mesma forma que nas análises com variação no tempo de comunicação, a variação da eficiência é bem menor quando comparada com a Aplicação 1, em virtude das características da aplicação fechada (figura 6.51). Essa variação permanece na ordem de 50%, porém a eficiência apresenta valores reduzidos da ordem de 20 a 30%. Por isso, para comparar com a Aplicação 1, a eficiência foi mostrada como estacionária na figura 6.51.

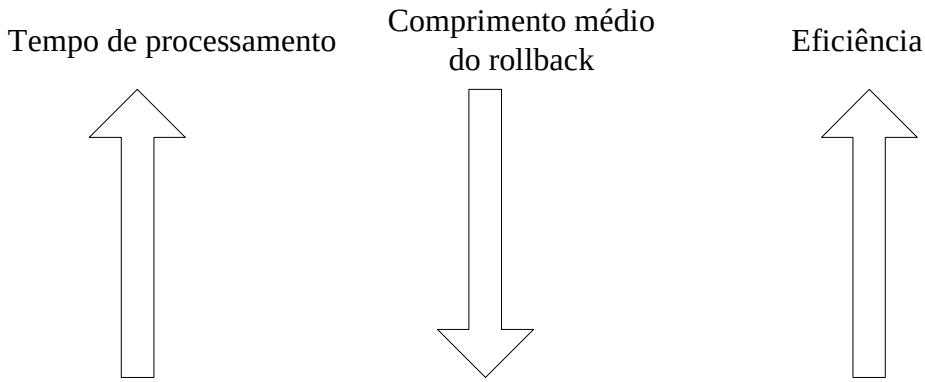


Figura 6.50: Representação da Relação entre Tempo de Processamento, Eficiência e Comprimento Médio do *Rollback Time Warp* Básico (Aplicação 1).

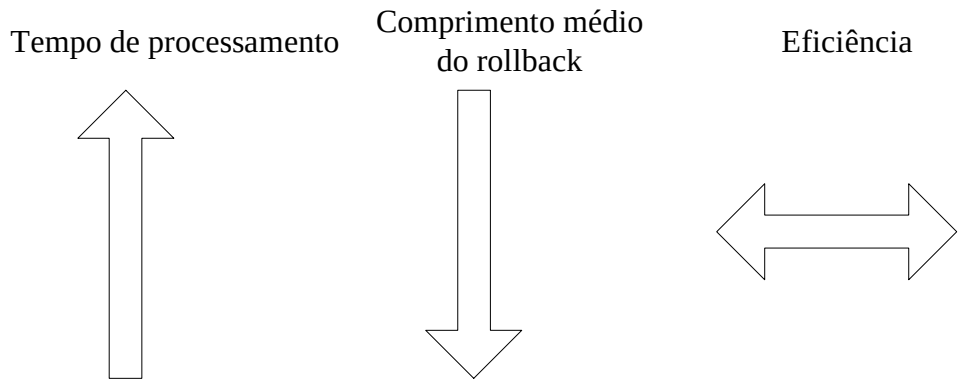


Figura 6.51: Representação da Relação entre Tempo de Processamento, Eficiência e Comprimento Médio do *Rollback Time Warp* Básico (Aplicação 2).

6.6 Influência dos Parâmetros da Aplicação

O objetivo desta seção é mostrar como o método proposto pode ser utilizado para analisar o comportamento da simulação distribuída sincronizada pelo protocolo *Time Warp* básico, de acordo com a variação nos parâmetros da aplicação que está sendo resolvida. Para o usuário da simulação distribuída, é interessante ter uma noção do comportamento da simulação frente a alterações nos parâmetros da aplicação necessários ao seu estudo de desempenho. Do mesmo modo, essa noção do comportamento pode ser utilizada no mecanismo de troca de protocolos de Morselli (Morselli 2000). As Aplicações 1 e 2 foram consideradas nessa análise.

6.6.1 Aplicação 1

Para mostrar a influência dos parâmetros da aplicação optou-se por analisar os parâmetros tempos entre-chegadas e porcentagem de tarefas que deixam o sistema.

6.6.1.1 Influência da Variação do Tempo Entre-Chegadas

O gráfico da figura 6.52 apresenta a variação da eficiência do protocolo *Time Warp* básico quando o tempo entre-chegadas é variado de 0,2 vezes até 2,0 vezes o tempo entre-chegadas original (processamento e comunicação são mantidos os valores padrões). Observa-se que a eficiência atinge seu ponto máximo com o tempo entre-chegadas padrão utilizado. Nas situações de aumento ou redução desse tempo, as marcas de tempo das mensagens correspondentes às chegadas dos novos clientes implicam na execução incorreta dos eventos, aumentando a ocorrência de *rollbacks* e, conseqüentemente, reduzindo a eficiência da simulação (as mensagens originárias do *plugin Gerador de Chegadas* e destinadas ao processo lógico 1 recebem o mesmo tratamento que uma mensagem originária do processo lógico 2 para o processo lógico 1).

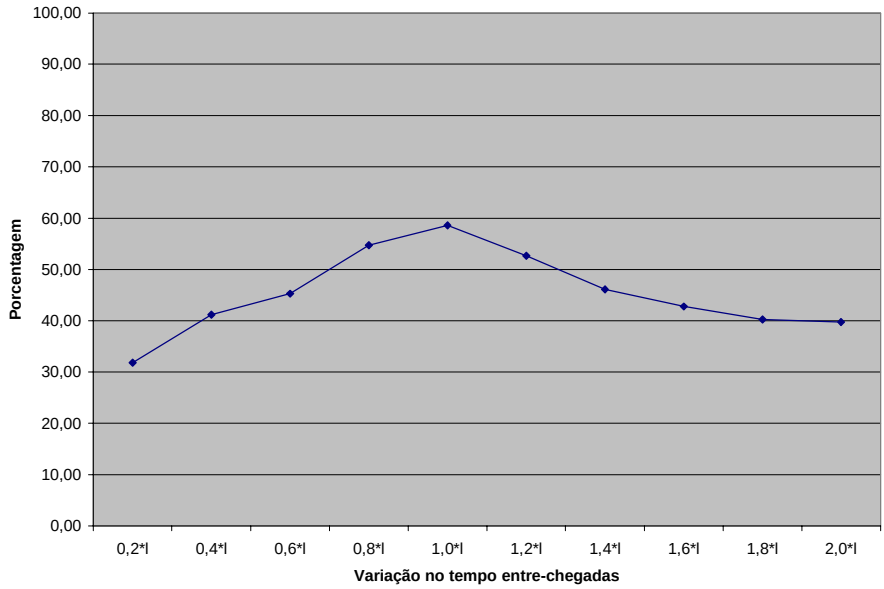


Gráfico 6.52: Influência da Variação do Tempo Entre-Chegadas na Eficiência do *Time Warp* Básico

6.6.1.2 Influência da Variação da Porcentagem de Tarefas que Deixam o Sistema

Um parâmetro importante de uma aplicação com realimentação é a porcentagem de clientes que deixam o sistema. No caso da simulação distribuída da Aplicação 1, quanto maior a porcentagem de clientes que deixam a aplicação, maior é o número de eventos executados no processo lógico 1, os quais não provocam o envio de mensagens para o processo lógico 2. Com isso, diminui a quantidade de mensagens que são enviadas do processo lógico 2 para o processo lógico 1 e, portanto, é menor a freqüência de *rollbacks* e, por isso, maior a eficiência (figura 6.53).

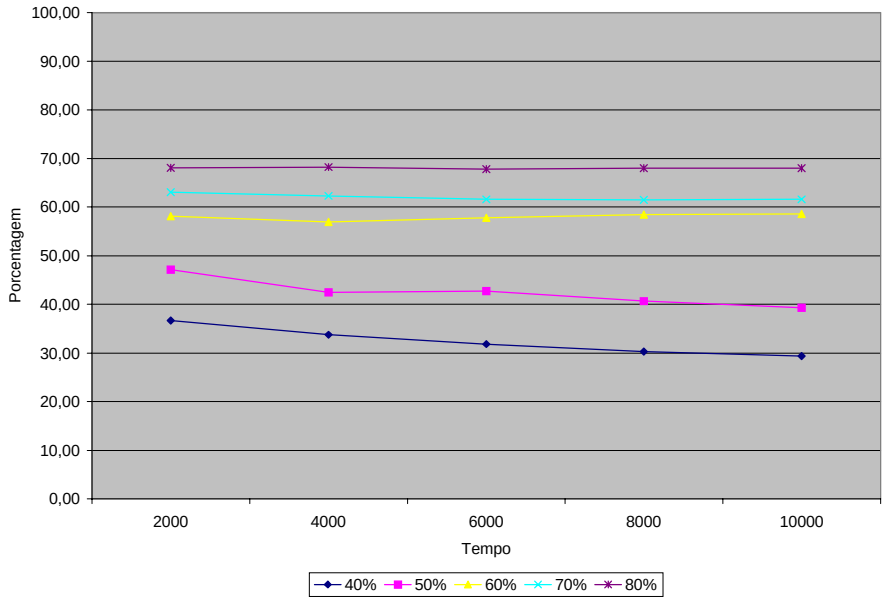


Figura 6.53: Influência da Variação da Porcentagem de Tarefas que Deixam o Sistema na Eficiência do *Time Warp* Básico (Aplicação 1).

6.6.2 Aplicação 2

O estudo de parâmetros da aplicação na Aplicação 2 envolveu a variação do número de tarefas inicialmente escalonadas para execução (a população no sistema fechado). Com apenas uma tarefa, não existe possibilidade de ocorrência de erro de causa e efeito e, portanto, a eficiência da simulação distribuída é de 100%. Com duas tarefas, a eficiência cai para 36% e com 3 tarefas, para 26% (figura 6.54). A taxa de queda da eficiência diminui com o aumento do número de tarefas, porque

implica no aumento do comprimento médio do *rollback*. Isso ocorre devido à característica da aplicação, que representa um sistema fechado. Nesse caso, as tarefas inicialmente escalonadas para execução, todas com tempo de ocorrência t=0,0 na aplicação, correspondem a eventos executados no processo lógico 1 com marcas de tempo iguais a 0,0. Todos esses eventos são executados pelo processo lógico 1, e geram novas mensagens para os processos lógicos 2 e 3 que, de forma independente, enviam novas mensagens para o processo lógico 1, contendo marcas de tempo que podem provocar *rollback* por mensagem nesse processo.

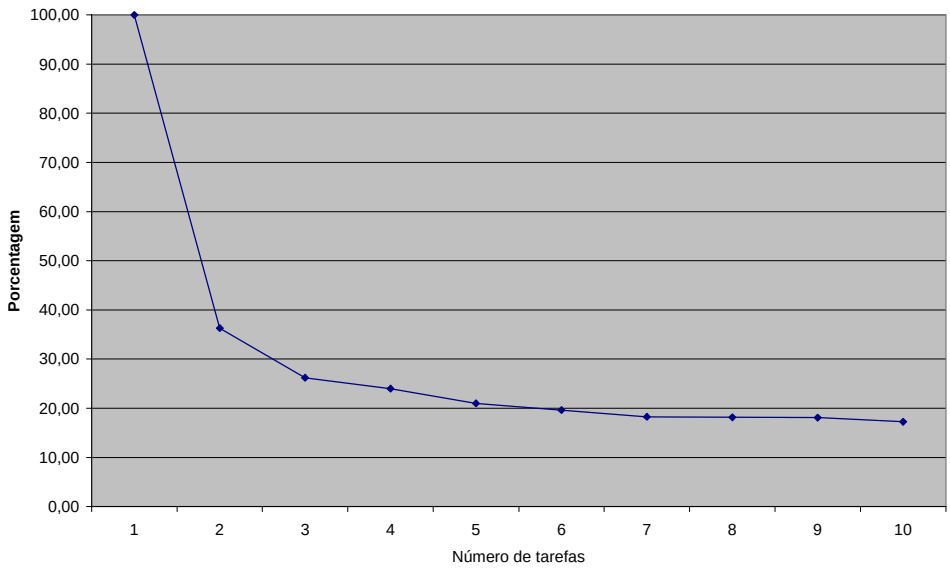


Figura 6.54: Influência da Variação do Número de Tarefas na Eficiência do *Time Warp* Básico (Aplicação 2).

6.7 Variação no Protocolo Básico

O protocolo *PDSP* (*Probabilistic Distributed discrete event Simulation Protocol)*, descrito no capítulo 3, efetua uma análise do comportamento da simulação distribuída para tentar prever, de forma probabilística, qual a marca de tempo esperada para o próximo evento. Dessa forma, decide se executa o evento da *Input Queue* ou se atrasa essa execução por um período de tempo, retornando em seguida para o início de suas atividades: analisar o *Buffer* de Entrada ou executar o próximo evento da *Input Queue*.

No capítulo 4 foi apresentado o *plugin* representando o protocolo *PDSP*. Nesta seção este *plugin* é utilizado para comparar o desempenho da sincronização do *Time Warp* básico com essa variante.

O objetivo desta seção é mostrar a aplicabilidade do método na comparação do desempenho entre protocolos otimistas, utilizando as Aplicações 1 e 2.

6.7.1 Aplicação 1

Os resultados obtidos mostram que o protocolo PDSP melhora a eficiência da simulação distribuída, quando comparado com o protocolo *Time Warp*, e que essa melhora depende da plataforma que está sendo utilizada.

A figuras 6.55 e 6.56 mostram, respectivamente, a eficiência, e o comprimento médio do *rollback*, comparando *Time Warp* básico e *PDSP*, em situações onde os tempos de processamento e comunicação são os padrões. Nos períodos de observação, o custo do algoritmo *PDSP* implica em menor número de eventos executados, porém a eficiência é melhor em cerca de 6%.

Observa-se que o tipo do protocolo também não influencia no valor médio do comprimento médio do *rollback* (figura 6.56). (a discussão sobre o comprimento médio do *rollback* foi efetuada na seção 6.4).

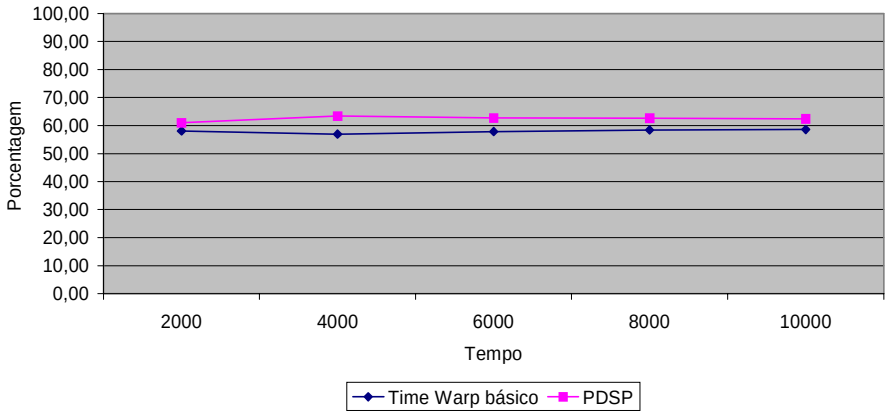


Figura 6.55: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento e Comunicação Padrão (Aplicação 1).

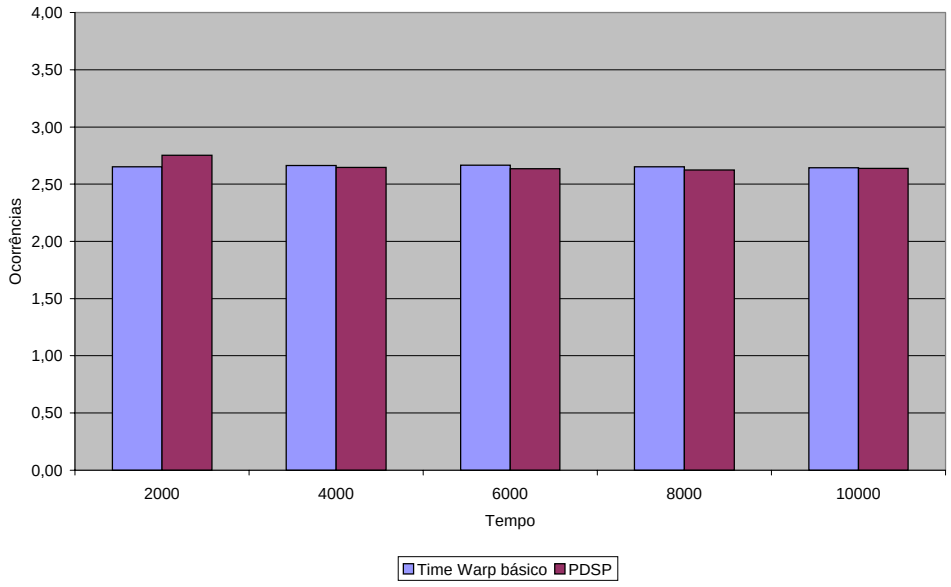


Figura 6.56: Comprimento Médio do *Rollback* no *Time Warp* Básico e no *PDSP* com Tempos de Processamento e Comunicação Padrão (Aplicação 1).

Quando o tempo de comunicação varia, o protocolo PDSP também apresenta melhor eficiência do que o protocolo *Time Warp*. Em todas as situações estudadas o melhor índice de aumento da eficiência foi em torno de 18% (figuras 6.57, 6.58, 6.59 e 6.60).

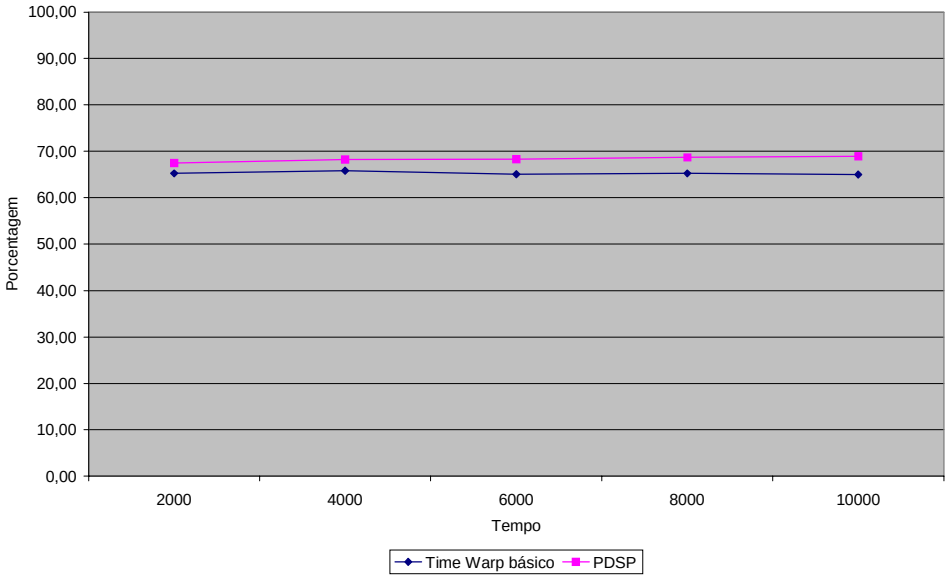


Figura 6.57: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Padrão e Tempo de Comunicação Dividido pela Metade (Aplicação 1).

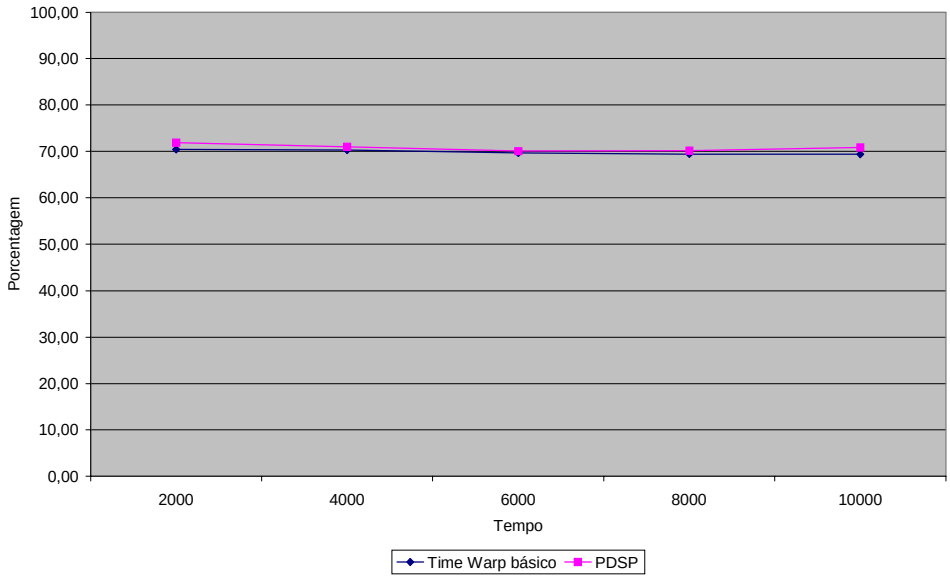


Figura 6.58: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Padrão e Tempo de Comunicação Igual a ¼ do Tempo Padrão (Aplicação 1).

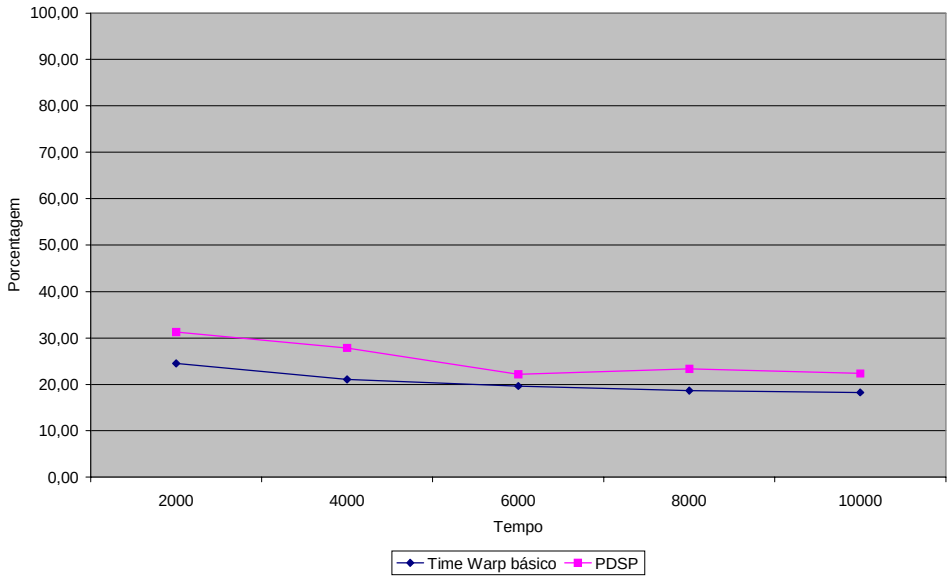


Figura 6.59: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Padrão e Tempo de Comunicação Igual a Uma Vez e Meia o Tempo Padrão (Aplicação 1).

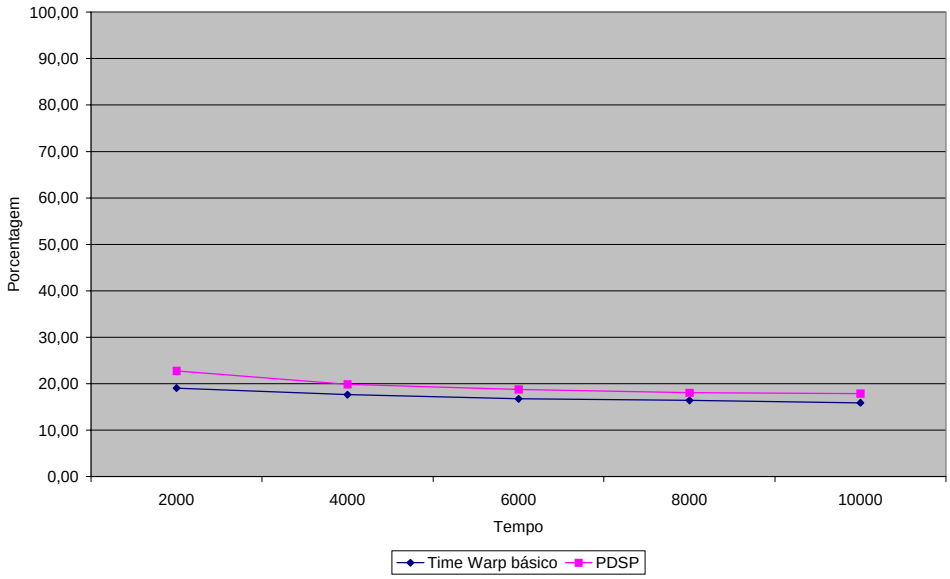


Figura 6.60: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Padrão e Tempo de Comunicação Igual a Duas Vezes o Tempo Padrão (Aplicação 1).

Também foram realizados experimentos variando-se o tempo de processamento e, em todos os casos, o protocolo *PDSP* apresentou melhor eficiência do que o *Time Warp* básico (figuras 6.61, 6.62, 6.63 e 6.64).

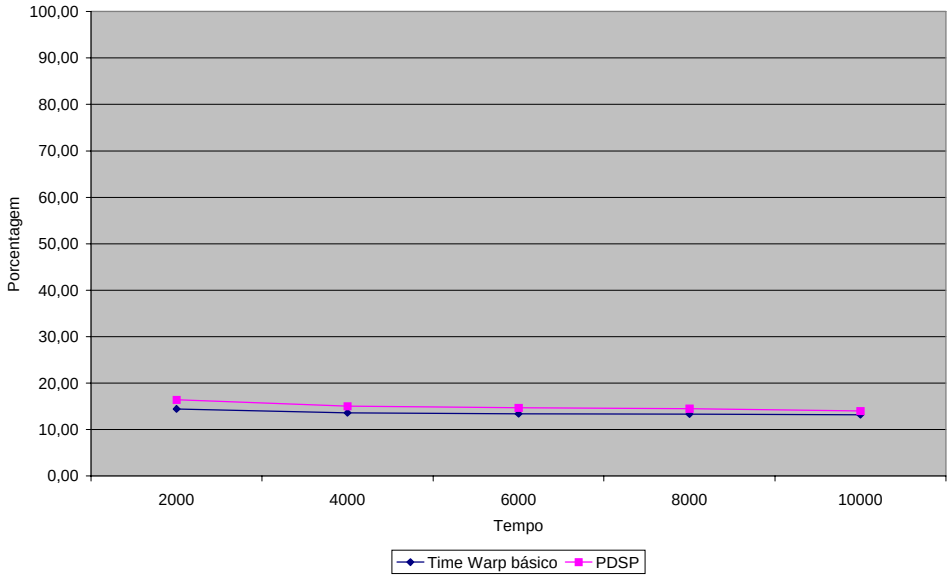


Figura 6.61: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Igual a ¼ do Tempo Padrão e Tempo de Comunicação Padrão (Aplicação 1).

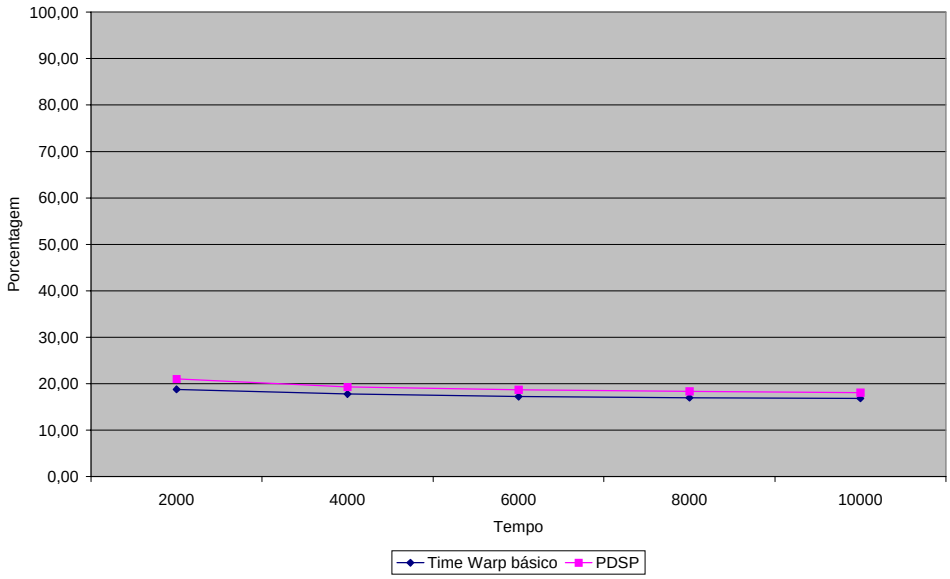


Figura 6.62: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Igual a Metade do Tempo Padrão e Tempo de Comunicação Padrão (Aplicação 1).

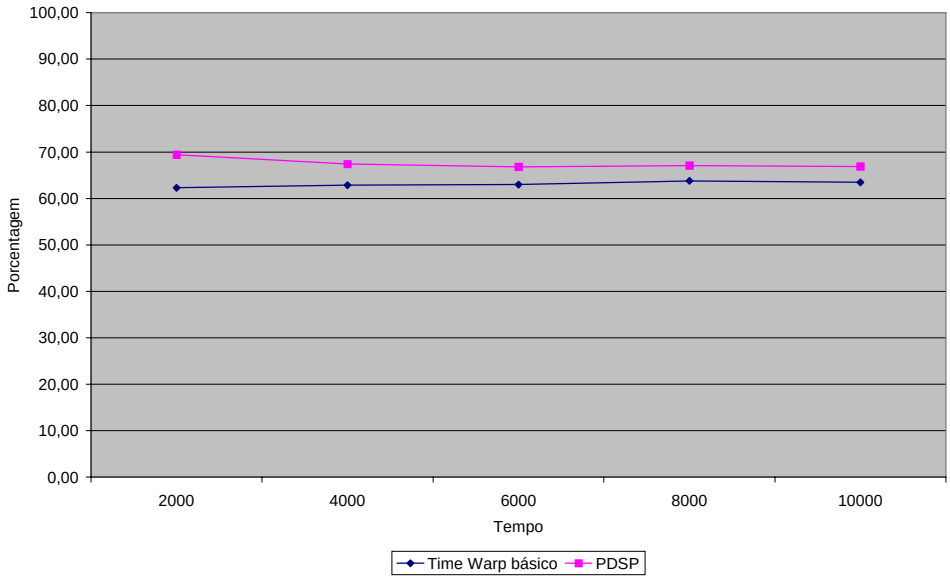


Figura 6.63: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Igual a Uma Vez e Meia o Tempo Padrão e Tempo de Comunicação Padrão (Aplicação 1).

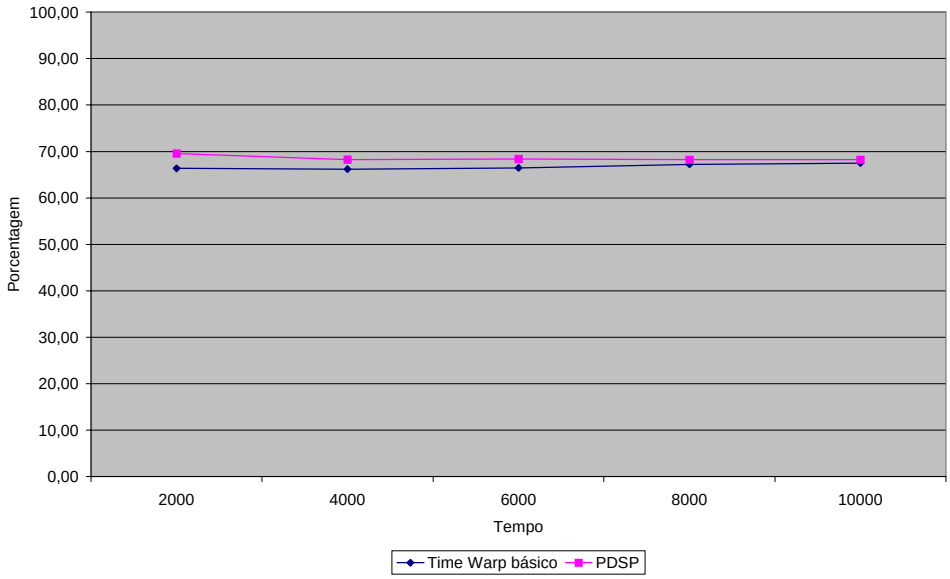


Figura 6.64: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Igual a Duas Vezes o Tempo Padrão e Tempo de Comunicação Padrão (Aplicação 1).

6.7.2 Aplicação 2

Com a Aplicação 2 observou-se o mesmo comportamento do protocolo PDSP em relação ao *Time Warp* básico observado com a Aplicação 1. O protocolo PDSP melhora a eficiência da simulação distribuída (figura 6.65), apesar de executar menos eventos no período observado. Isso acontece por causa do tempo gasto no bloqueio do processo lógico. Porém, o protocolo probabilístico consegue executar um número maior de eventos corretos, com menos *rollbacks*. O comprimento médio do *rollback* não sofre alteração (figura 6.66).

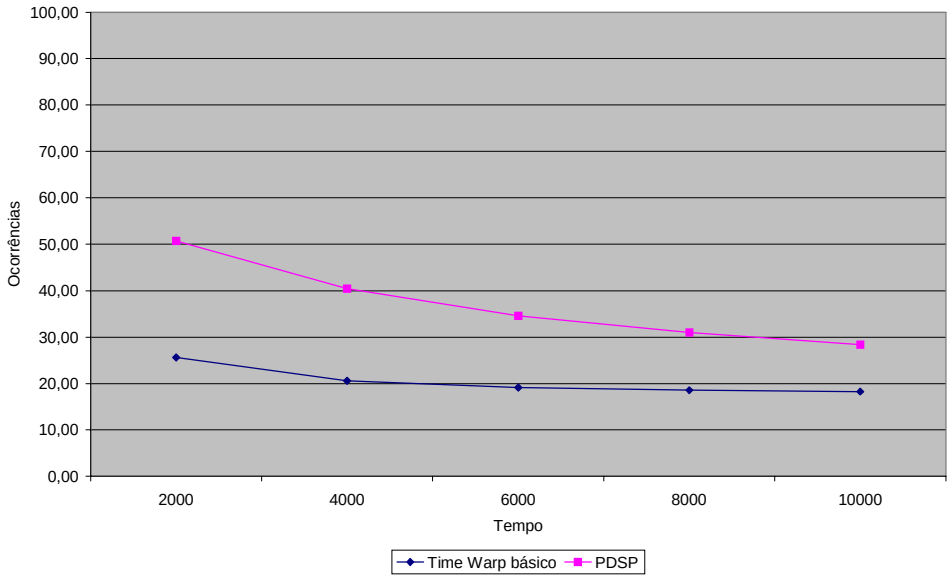


Figura 6.65: Comparando Eficiência do *Time Warp* Básico e do PDSP com Tempos de Processamento e Comunicação Padrão (Aplicação 2).

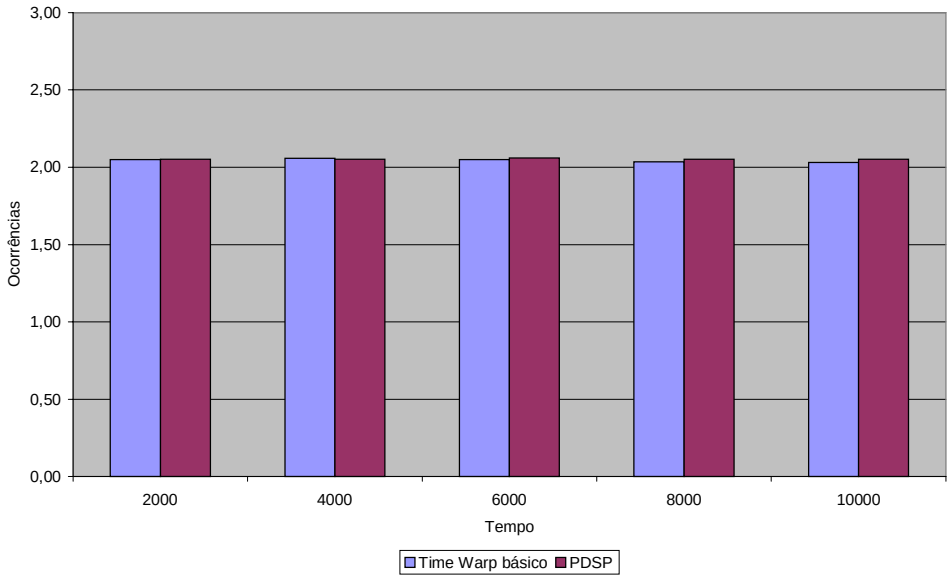


Figura 6.66: Comprimento Médio do *Rollback* no *Time Warp* e no *PDSP* com Tempos de Processamento e Comunicação Padrão (Aplicação 2).

Com o tempo de comunicação menor do que o padrão observa-se que a eficiência do protocolo *PDSP* é maior do que a do *Time Warp* básico, porém com a comunicação mais lenta praticamente não existe diferença entre a eficiência obtida pelos dois protocolos. Isso ocorre devido ao fato de que a comunicação torna-se

muito lenta e o limitante do desempenho é o tempo de processamento que é igual em todos os experimentos (figuras 6.67, 6.68, 6.69 e 6.70).

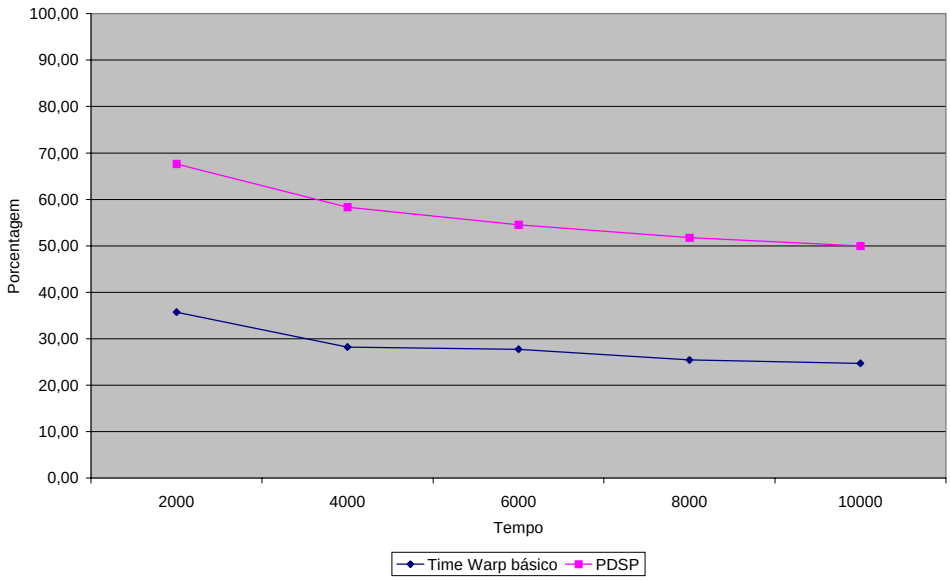


Figura 6.67: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Padrão e Tempo de Comunicação Dividido pela Metade (Aplicação 2).

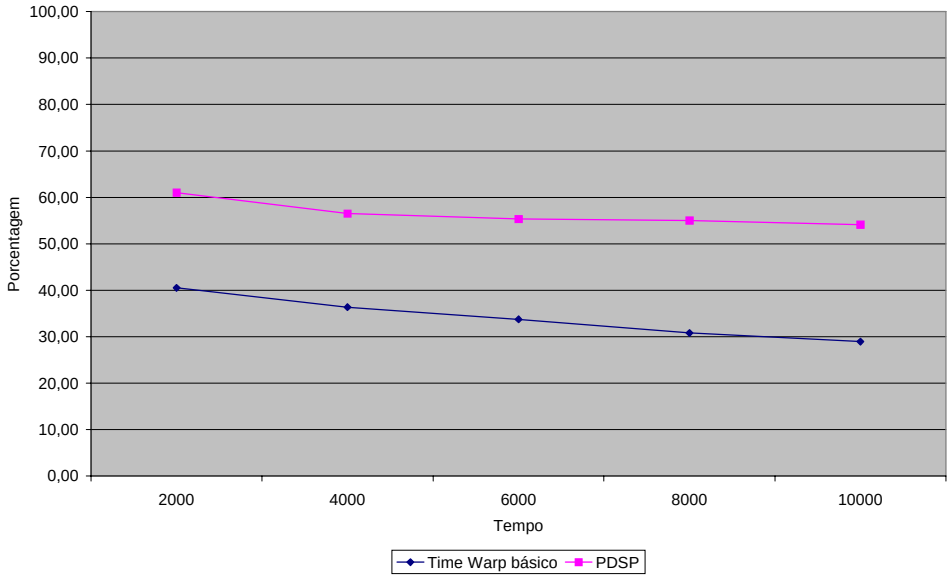


Figura 6.68: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Padrão e Tempo de Comunicação Igual a ¼ do Padrão (Aplicação 2).

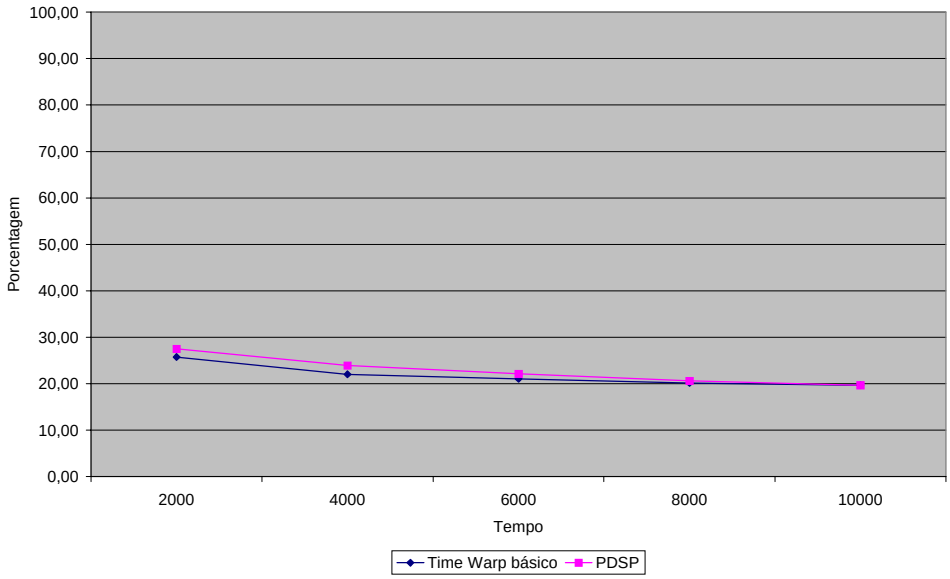


Figura 6.69: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Padrão e Tempo de Comunicação Igual a Uma Vez e Meia o Tempo Padrão (Aplicação 2).

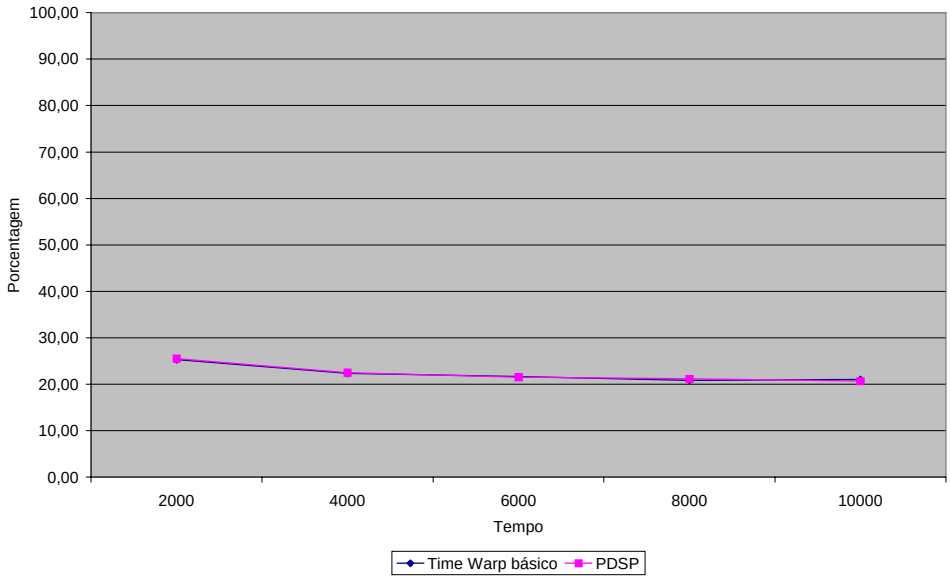


Figura 6.70: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempos de Processamento Padrão e Tempo de Comunicação Igual a Duas Vezes o Tempo Padrão (Aplicação 2).

Também foram realizados experimentos variando-se o tempo de processamento e, em todos os casos, o protocolo *PDSP* apresentou melhor eficiência

do que o *Time Warp* básico. O melhor caso, dos testes realizados, apresentado na figura 6.71, obteve melhora na eficiência da ordem de 64%.

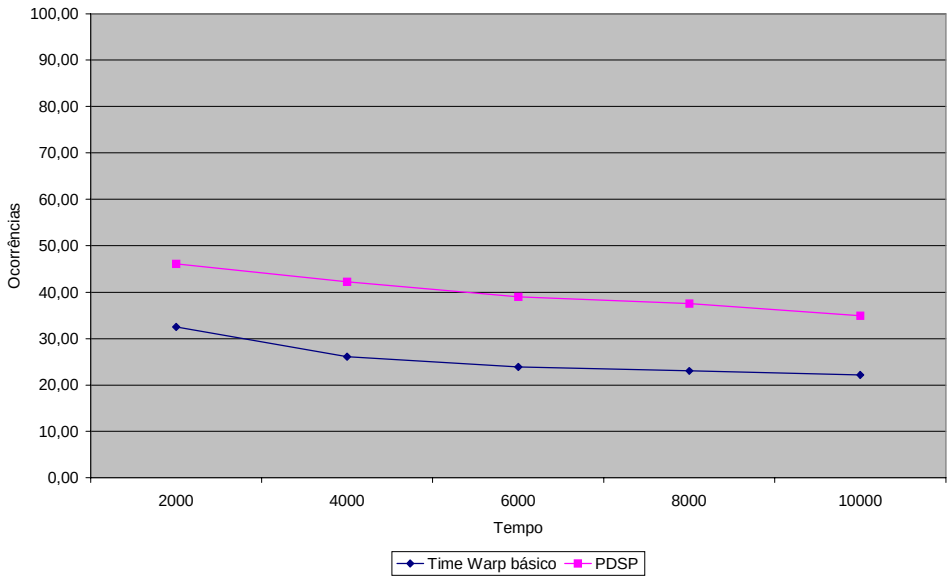


Figura 6.71: Comparando Eficiência do *Time Warp* Básico e do *PDSP* com Tempo de Processamento Padrão Igual a Uma Vez e Meia o Tempo Padrão e Tempo de Comunicação Padrão (Aplicação 2).

6.8 Previsão do *Speedup*

O tempo da simulação do modelo do protocolo de sincronização, na ferramenta *ALPHA/Sim* (relógio da simulação) corresponde a uma estimativa do tempo de execução da simulação distribuída na plataforma considerada. Isso possibilita a verificação do *speedup* de forma aproximada.

O *speedup* é uma das métricas mais utilizadas para determinar se a utilização do processamento paralelo ou distribuído está sendo vantajoso e quantificar o desempenho alcançado. É definido como o aumento de velocidade observado em um computador paralelo com *p* elementos de processamento, em relação a um computador seqüencial, como apresentado pela equação 6.3 (Hwang; Xu 1998).

Equação 6.3

$Sp = Ts/Tp$

Onde:

Sp = *speedup* observado em um computador com *p* elementos de processamento;
Ts = tempo de execução do programa em um único elemento de processamento;
Tp = tempo de execução do mesmo programa em um computador paralelo, com *p* elementos de processamento.

O método de avaliação proposto no capítulo 5 permite ao usuário utilizar a métrica *speedup* para verificar a tendência do comportamento da simulação distribuída em relação ao tempo de execução, quando comparado com a simulação seqüencial. A figura 6.72 ilustra as definições de *Tp* e *Ts* utilizadas neste trabalho. A porção de tempo que cada processo lógico utiliza para executar eventos na ordem cronológica correta corresponde ao tempo necessário para executar parte dos eventos da simulação seqüencial (*Tsi*). A somatória dos tempos correspondentes a todos os processos lógicos fornece uma previsão do tempo da simulação seqüencial. O tempo da simulação distribuída corresponde ao tempo que os processos lógicos utilizam para executar eventos (corretos ou não) mais o tempo de comunicação e o tempo das tarefas de sincronização do protocolo, executadas em cada processo lógico (salvamento de estados, manipulação de estruturas de dados e eventos executados de maneira errada (*Ti*) e eventos executados corretamente (*Tsi*)).

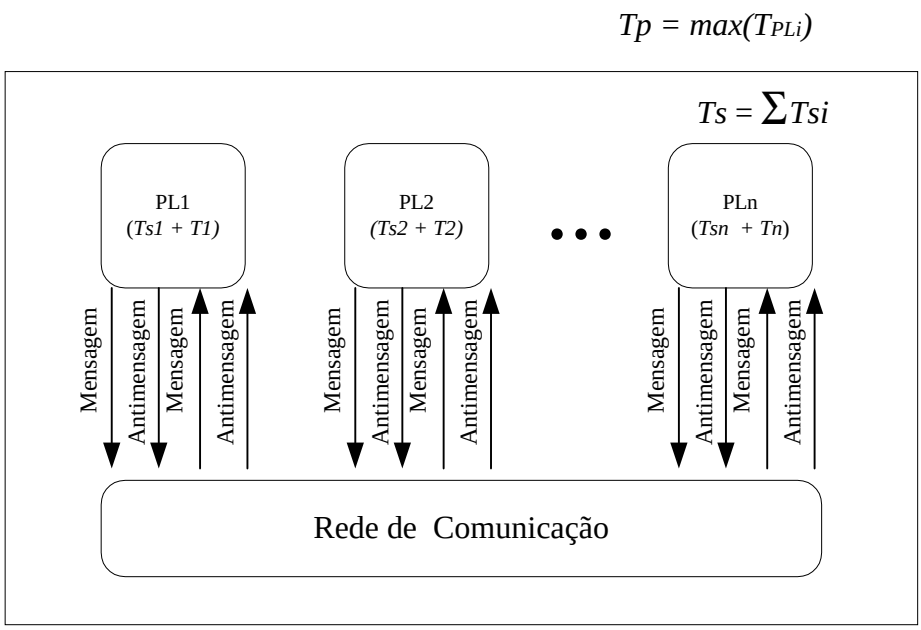


Figura 6.72: Identificação de *Tp* e *Ts* para o *Speedup* Teórico Utilizado no Método de Avaliação.

A tabela 6.2 apresenta o resultado obtido com a Aplicação 1, em situações de teste considerando variações do tempo de processamento e do tempo de comunicação (tempo de processamento e tempo de comunicação padrões – P-C, tempo de processamento padrão e tempo de comunicação reduzido à metade – P-0,5C, e tempo de processamento padrão e tempo de comunicação reduzido a um quarto do valor padrão – P-0,25C). O *speedup* é menor do que 1 devido ao fato de que a aplicação considerada (Aplicação 1) representa um modelo com pequeno número de processos lógicos e existe sobrecarga imposta pelo protocolo e pelo meio de comunicação. Essa aplicação da métrica *speedup* mostra que o método de avaliação proposto no capítulo 5 é adequado também para se ter uma previsão do *speedup* da simulação distribuída, antes da sua implementação ser efetivada.

Tabela 6.2: Valores de *Speedup* Observados.

	P-C	P-0,5C	P-0,25C
Tp	60000 ms	60000 ms	60000 ms
Ts	38742,56 ms	44791,64 ms	46759,71 ms
Sp	0,64	0,75	0,78

6.9 Considerações Finais

Este capítulo mostrou como utilizar o método de avaliação detalhado no capítulo 5, apresentando como estudo de caso a avaliação dos protocolos *Time Warp* básico e *PDSP*, considerando diversas situações práticas.

As contribuições deste capítulo são:

- A discussão de resultados de desempenho do protocolo *Time Warp* básico, em diferentes situações de variação dos parâmetros da plataforma, da aplicação e resultados de desempenho comparando o *Time Warp* básico com o protocolo *PDSP*, porque permitiu mostrar que o método de avaliação é viável para estudar o comportamento de um protocolo otimista no tempo da simulação distribuída, quando existe variação dos parâmetros e também para comparar protocolos através do uso dos *plugins*;
- A identificação de situações onde o mecanismo proposto em (Morselli 2000) para efetuar a troca de protocolos de sincronização em tempo de execução pode ser efetivamente aplicado, em função da estabilização do índice de eficiência dos protocolos otimistas observados. Essas situações envolvem ambas as aplicações utilizadas nos estudos de caso pois os gráficos de eficiência mostraram que pode-se aplicar o referido mecanismo para passar a sincronizar a simulação distribuída através do protocolo *CMB*;
- A definição de como construir e aplicar a métrica *speedup* aproximado, para se obter um valor estimativo do comportamento da simulação distribuída otimista em relação à simulação seqüencial;
- Os resultados de desempenho sumarizados na tabela 6.3.

Tabela 6.3: Resultados de Desempenho.

Comportamento no Tempo da Simulação Distribuída (*Time Warp* básico)
A eficiência da simulação distribuída se mantém constante na aplicação com características de sistema misto, porém se reduz na aplicação representando sistema fechado; o comprimento médio do *rollback* é estável no tempo, independentemente do tipo da aplicação.
Influência dos Parâmetros da Plataforma
Plataforma com tempo de comunicação mais lento provoca queda acentuada na eficiência da Aplicação 1 e queda menos acentuada na Aplicação 2. Plataforma com tempo de processamento mais lento provoca aumento de eficiência na Aplicação 1 e queda pouco acentuada na Aplicação 2. A influência da plataforma depende também das características da aplicação. Um guia de referência foi apresentado, mostrando como obter determinada eficiência, para a Aplicação 1.
Influência dos Parâmetros da Aplicação
A influência na variação da eficiência é importante para permitir ao usuário ter noção do desempenho que vai obter e também auxiliar na tomada de decisão para utilização do mecanismo de troca de protocolos de Morselli (Morselli 2000).
Influência da Variação do Protocolo de Sincronização
O comprimento médio do *rollback* permanece constante no tempo da simulação distribuída, independentemente do tipo do protocolo de sincronização utilizado. O ganho de eficiência do protocolo probabilístico depende das características da plataforma e da aplicação.

Capítulo 7

Conclusões, Contribuições e Propostas para Trabalhos Futuros

Este capítulo apresenta as conclusões e contribuições resultantes do trabalho desenvolvido e as propostas para a continuidade do trabalho.

7.1 Conclusões Gerais

Esta tese propôs um método para avaliação de desempenho de protocolos de sincronização otimistas baseados no *Time Warp*. Parte-se da premissa de que o usuário já optou pela utilização da simulação distribuída e pelos protocolos otimistas. Dessa forma, o método será utilizado para definir qual protocolo otimista é o mais adequado. No decorrer do trabalho mostrou-se a viabilidade de se utilizar a simulação como técnica de avaliação de protocolos de sincronização, através da construção de um modelo básico de um processo lógico *Time Warp*. Esse modelo básico pode ser alterado através de *plugins*, que inserem e/ou modificam as características do protocolo, resultando em modelos de protocolos variantes.

O método proposto nesta tese é inovador sob diversos aspectos, destacando-se por permitir uma fácil caracterização de diferentes arquiteturas e protocolos otimistas a

serem avaliados, fornecendo, dessa forma, uma previsão do comportamento da simulação distribuída, sem a necessidade de implementação da mesma. Definir qual é o melhor protocolo de sincronização não é uma tarefa trivial, devendo-se levar em consideração as características da aplicação que vai ser resolvida pela simulação distribuída, as características da plataforma e também do próprio protocolo de sincronização.

Para que o modelo básico fosse construído e a aplicabilidade dos *plugins* pudesse ser demonstrada (como no caso do mecanismo de salvamento de estados), foi necessário organizar os protocolos variantes do *Time Warp* em uma nova taxonomia, proposta no capítulo 3. A análise da bibliografia demonstra que as poucas taxonomias existentes têm abrangência limitada, não facilitando dessa forma a tarefa de escolha do protocolo de sincronização mais adequado a ser empregado. A nova taxonomia proposta procurou corrigir essas falhas e fornecer o suporte necessário para o desenvolvimento do método proposto.

A taxonomia proposta no capítulo 3 foi necessária para a identificação das características do *Time Warp* básico que deveriam ser representadas em um modelo e a criação dos domínios de *plugins*. Justifica-se assim a necessidade de um capítulo contendo parte de revisão bibliográfica (as classificações de Das e Srinivasan) necessária à apresentação da nova taxonomia. Dessa forma, o princípio básico do desenvolvimento do modelo de um processo lógico *Time Warp* seguiu a construção da taxonomia, de forma hierárquica e modular (através da conexão dos *plugins*).

Além da revisão sobre o estado da arte, o estudo permitiu a identificação das características do protocolo *Time Warp* que são importantes para a avaliação de desempenho. Essas características envolvem a parte de recepção e tratamento de mensagens e antimensagens; a execução de eventos, que pode levar ao escalonamento local de novos eventos ou o envio de mensagens a processos lógicos remotos e também os sub-algoritmos, importantes para o desempenho do *Time Warp*, que envolvem o salvamento de estados, o cálculo do *GVT*, *Fossil Collection*, o cancelamento de mensagens e a recuperação de erros de causa de efeito.

Identificadas as características importantes, representou-se o comportamento de um processo lógico *Time Warp* básico. Os algoritmos utilizados na literatura para cada

uma das características descritas foram explicados no capítulo 3 e no capítulo 4 mostrou-se como esses algoritmos podem ser aplicados para criar modelos representativos de processos lógicos de variantes do *Time Warp*.

Uma vez definido o modelo, o passo seguinte foi a escolha de uma técnica para representá-lo. Existem diferentes técnicas para a descrição de modelos, entre as quais as Redes de Fila, as Redes de Petri e os *Statecharts*. Cada uma dessas técnicas tem as suas vantagens e desvantagens. Optou-se por Redes de Petri devido à sua flexiblidade e facilidade para representar concorrência, fundamental na descrição do comportamento de um processo. As Redes de Fila não se mostraram adequadas para a representação do comportamento de um processo lógico devido ao fato de não permitirem a representação de detalhes e nem concorrência. *Statecharts* também apresentam limitações. *ALPHA/Sim*, uma ferramenta para descrição e simulação de modelos representados através de Redes de Petri, foi utilizada por permitir o uso das características das Redes de Petri Estocásticas Generalizadas, Coloridas e Hierárquicas.

Os resultados de desempenho discutidos no capítulo 6 demonstraram que o método de avaliação pode ser utilizado no estudo do comportamento da simulação distribuída no tempo, na análise comparativa entre protocolos otimistas, no estudo do desempenho em arquiteturas distintas e, também, no estudo do impacto dos parâmetros da aplicação no desempenho da simulação distribuída (nesse caso o usuário obtém um conhecimento prévio de como vai se comportar a sua simulação) e na previsão de estimativa do *speedup* da simulação distribuída. O método de avaliação também é útil nas situações onde o usuário está convencido de que a simulação distribuída sincronizada por um determinado protocolo otimista é a melhor solução para resolver a sua aplicação. Nesse caso, o método permite que se faça uma avaliação do comportamento da simulação distribuída em relação à granulosidade dos eventos e à divisão da aplicação em processos lógicos, antes da implementação ser efetivada.

7.2 Contribuições deste Trabalho

O trabalho descrito nesta tese aborda vários aspectos relacionados aos protocolos de sincronização otimista em simulação distribuída. A contribuição fundamental desta

tese é a concepção de um método inovador para avaliar os protocolos de sincronização otimistas baseados em variações do protocolo *Time Warp* básico. O método é inovador no sentido de que utiliza a técnica de modelagem (Redes de Petri) para construir um modelo básico, a partir do qual diferentes variantes podem ser facilmente estudadas por meio da inserção de pequenos modelos, que caracterizam os *plugins*. O método permite a exploração das características tanto da simulação distribuída, como da plataforma computacional utilizada, da rede de interconexão e da aplicação a ser resolvida. Além disso, uma estimativa do *speedup* da simulação distribuída também pode ser obtida.

Os passos necessários para efetuar o estudo são simples e objetivos, facilitados pelo fato de que o usuário da simulação distribuída só tem que configurar os blocos básicos de processos lógicos, definindo a sua interconexão, e fornecer os parâmetros necessários. O encapsulamento dos processos lógicos em blocos e o uso de *plugins* retira do usuário a responsabilidade de definir modelos em Redes de Petri. A única dificuldade ainda existente é o ambiente de simulação utilizado, que exige um esforço computacional ainda longo, mas que poderá ser otimizado em futuras utilizações ou adotando-se outros ambientes de simulação de Redes de Petri, ou ainda uma plataforma computacional mais adequada.

O método utilizado propõe o emprego de meta-simulações, o que não constitui um experimento muito fácil de ser entendido, mas a sistematização proposta nesta tese elimina possíveis dúvidas e conduz os usuários a fornecer as informações necessárias ao estudo de forma consistente.

O capítulo 6 desta tese mostra a potencialidade do método desenvolvido, aplicando-o para o *Time Warp* básico e para o protocolo PDSP (através do uso de um *plugin*). O estudo apresentado no capítulo 6, por si, apresenta uma série de contribuições para a área de simulação distribuída, identificando características relacionadas ao comportamento e ao impacto dos *rollbacks* no desempenho da simulação distribuída, a tendência do comportamento da simulação distribuída no tempo, a influência dos parâmetros da plataforma e da aplicação na eficiência da simulação distribuída, o desempenho do *Time Warp* básico comparado ao desempenho

do protocolo PDSP e também, em alguns casos, comprovando diversos resultados mostrados na literatura para outras situações.

De modo geral, pode-se ainda identificar as seguintes contribuições oriundas de todo o trabalho desenvolvido nesta tese:

- Análise crítica da bibliografia existente, fornecendo um contexto geral dos protocolos de sincronização que limitam o otimismo do protocolo *Time Warp* básico;
- Levantamento sobre as taxonomias discutidas na literatura e identificação de divergências e pontos falhos;
- Proposta de uma nova taxonomia, concisa e hierárquica, que classifica os protocolos de sincronização otimistas de acordo com as características de implementação, características de sincronização e modo de limitar o otimismo;
- Identificação das características do protocolo de sincronização *Time Warp* básico que são importantes para a avaliação de desempenho;
- Desenvolvimento de um modelo em Redes de Petri representando um processo lógico *Time Warp* básico e extensão desse modelo, através de *plugins*, para outros protocolos otimistas variantes do *Time Warp*;
- Definição de domínios de *plugins*, viabilizado pela proposta de nova taxonomia, através dos quais modelos das variantes do *Time Warp* podem ser construídos;
- Uma descrição detalhada, textual, do comportamento de um processo lógico *Time Warp* também foi produzida na fase de confecção do modelo, a qual pode ser utilizada como base para futuras implementações;
- O método de avaliação proposto, que permite que o usuário faça um estudo com os diversos protocolos otimistas, utilizando os domínios de *plugins*, para simular o comportamento da simulação distribuída, antes da sua implementação. Isso mostra a viabilidade de se aplicar a simulação para estudar protocolos de sincronização em simulação distribuída;

- A possibilidade de variar as características da plataforma, que permite que o usuário da simulação distribuída verifique o comportamento em plataformas diferentes da que possui, ou ainda procure identificar uma plataforma que melhor se adapta às suas necessidades;
- Estudo de desempenho do protocolo *Time Warp* básico, que permitiu identificar, por exemplo, que o comprimento médio do *rollback* permanece constante no tempo da simulação distribuída;
- Estudo de desempenho comparando o protocolo *Time Warp* básico e o protocolo *PDSP*, mostrando que o segundo apresenta melhor eficiência em relação ao primeiro, para as aplicações utilizadas.

7.3 Propostas para Trabalhos Futuros

O trabalho desenvolvido nesta tese poderá ter continuidade considerando:

- O desenvolvimento de um ambiente automático que permita a descrição gráfica da aplicação, o particionamento em processos lógicos, a parametrização e a obtenção das métricas;
- A inclusão de características de balanceamento de carga para estudar os efeitos da divisão em processos lógicos no desempenho da simulação distribuída;
- A análise da viabilidade de utilização de novas técnicas e ferramentas, como *Estelle* e também *Statecharts* (Francês 2001). Dessa forma, pode-se evitar a explosão de lugares e transições que podem ocorrer quando o nível de detalhamento do modelo for aprofundado;
- A extensão do método de avaliação proposto neste trabalho para avaliar também protocolos conservativos, fornecendo ao usuário a opção de poder verificar qual das abordagens irá apresentar melhor desempenho ao resolver sua aplicação;
- O desenvolvimento dos *plugins* dos demais protocolos variantes do *Time Warp* básico;

- A utilização do método para estudar a porcentagem de tempo utilizada pelos processos lógicos nas tarefas envolvidas com os *rollbacks*. Isso permitirá a identificação de pontos críticos e a utilização do método para estudar a viabilidade de novos protocolos de sincronização, bastando desenvolver os *plugins* necessários, sem a necessidade de implementá-los;
- O aprofundamento do estudo de desempenho dos protocolos otimistas utilizando o método proposto.

Referências Bibliográficas

ALLEYNE, P.; TROPPER, C. On the Parallel Simulation of Fixed Channel Allocation Algorithms. *Mobile Networks and Applications*, p. 209-218, v.5, n. 3, 2000.

ALPHATECH CORPORATION. ALPHA/Sim User's Guide, 1995.

ARVIND, D. K.; SMART, C. R. Hierarchical Parallel Discrete Event Simulation in Composite ELSA**.** In: *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS'92),* p. 147-156, 1992.

AYANI, R.; RAJAEI, H. Parallel Simulation using Conservative Time Windows. In: *Proceedings of the 1992 Winter Simulation Conference*, p. 709-717, 1992.

AYANI, R.; RAJAEI, H. Parallel Simulation Based on Conservative Time Windows: a Performance Study. *Concurrency: Practice and Experience*, v. 6, n° 2, p. 119-142, 1994.

BAGRODIA, R. L. Perils and Pitfals of Parallel Discrete-Event Simulation. In: *Proceedings of the 1996 Winter Simulation Conference (WSC'96),* p. 136-146, 1996.

BAGRODIA, R. L.; MEYER, R.; TAKAI, M.; CHEN, Y.; ZENG, X.; MARTIN, J.; SONG, H. Y. PARSEC: A Parallel Simulation Environment for Complex Systems. *IEEE Computer*, v. 3, n. 10, p. 77-85, 1998.

BALAKRISHNAN, V.; FREY, P.; ABU-GHAZALEH, N.; WILSEY, P. A. A Framework for Performance Analysis of Parallel Discrete Event Simulators. In: *Proceedings of the 1997 Winter Simulation Conference*, p. 429-436, 1997.

BALL, D.; HOYT, S. The Adaptive Time Warp Concurrency Control Algorithm. *Simulation Series*, v. 22, n. 11, p. 174-177. The Society for Computer Simulation, 1990.

BELLENOT, S. Performance of a Riskfree Time Warp Operating System. In: *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS 93)*, p. 155-158, 1993.

BRYANT, R. E. *Simulation of Packet Communications Architecture Computer Systems.* Technical Report, MIT-LCS-TR-188, Massachussets Institute of Technology, 1977.

BUDKOWSKI, S.; DEMBINSKI, P. An Introduction to Estelle: A Specification Language for Distributed Systems. *Computer Network and ISDN Systems*, 14, p.3-23, 1987.

CAROTHERS, C.; PERUMALLA, K. S.; FUJIMOTO, R. M. Efficient Optimistic Parallel Simulations using Reverse Computing. In: *Proceedings of The 13th Workshop on Parallel and Distributed Simulation* (PADS'99), p.126-135,

1999a.

CAROTHERS, C.; PERUMALLA, K. S.; FUJIMOTO, R. M. The Effect of State-Saving in Optimistic Simulation on a Cache-Coherent Non-Uniform Memory Access Architecture. In: *Proceedings of the 1999 Winter Simulation Conference (WSC'99)*, p. 1624-1633, 1999b.

CAROTHERS, C. D.; BAUER, D.; PEARCE, S. ROSS: A High-Performance, Low Memory, Modular Time Warp System. In: *Proceedings of The 14th Workshop on Parallel and Distributed Simulation (PADS'2000*), p. 53-60, 2000.

CHETLUR, M.; N. ABU-GHAZALEH; R. RADHAKRISHNAN; WILSEY, P. A. Optimizing Communication in Time Warp Simulators. In: *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98)* (Banff, Canada, May 26-29), p. 64-91, 1998.

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, 1979.

CHIOLA, G.; FERSCHA, A. Performance Comparison of Distributed Petri Net Simulations. In: *Proceedings of the 1995 Summer Simulation Conference*, p. 24-26, 1995.

CORTELESSA, V.; QUAGLIA, F. Aggressiveness/Risk Effects based Scheduling in Time Warp. In: *Proceedings of the 2000 Winter Simulation Conference*, p. 409-417, 2000.

DAS, S. R. Adaptive Protocols for Parallel Discrete Event Simulations. In: *Proceedings of the 1996 Winter Simulation Conference (WSC'96)*, p. 186-193, 1996.

DITTRICH, G. Modeling of Complex Systems Using Hierarchical Petri Nets. *Codesign - Computer-Aided Software / Hardware Engineering*, IEEE Press, p. 128-144, 1995.

DJEMAME, K.; BETTAZ, M.; GILLES, D. C.; MACKENZIE, L. M. Performance Comparison of High Level Algebraic Nets Distributed Simulation Protocols. In: *Proceedings of the 1996 Winter Simulation Conference (WSC'96)*, p. 621-628, 1996.

D'SOUZA, L. M.; FAN, X.; WILSEY, P. Modifications to the pGVT Algorithm to Eliminate Acknowledgement Messages and Improve the GVT Broadcast Frequency. In: *Proceedings of the World Congress on Systems Simulation*, 1997.

ELMAGHRABY, A. S.; ELFAYOUMY, S. A.; KARACHIWALA, I. S.; GRAHAM, J. H.; EMAM, A. Z.; SLEEM, A. M. Web-based Performance Visualization of Distributed Discrete Event Simulation. In: *Proceedings of the 1999 Winter Simulation Conference (WSC'99)*, p. 1618-1623, 1999.

FABBRI, A. GVT and Scheduling in Space Time Memory Based Techniques. In: *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS'99)*, p. 54-61, 1999.

FABBRI, A.; DONATIELLO, L. SQTW: a Mechanism for State-dependent Parallel Simulation. Description and Experimental Study. In: *Proceedings of the 11th Workshop on Parallel and Distributed Simulation (PADS'97)*, p.82-88,

1997.

FERSCHA, A.; CHIOLA, G. Self-Adaptive Logical Processes: the Probabilistic Distributed Simulation Protocol. In: *Proceedings of the 27th Annual Simulation Symposium*, p. 78-88. IEEE Computer Society Press, 1994.

FERSCHA, A. Probabilistic Adaptive Direct Optimism Control in Time Warp. In: *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS'95)***,** p. 120-129, IEEE Computer Society Press, 1995a.

FERSCHA, A. Adaptive Model Parallelism Exploitation in Parallel Simulation. In: *Proceedings of EUROSIM'95*, p. 241-248, 1995b.

FERSCHA, A. Parallel and Distributed Simulation of Discrete Event Systems. In: *Parallel and Distributed Computing Handbook*. Chapter 35. McGraw-Hill, 1995c.

FERSCHA, A. Estimating Rollback Overhead for Optimism Control in Time Warp. In: *Proceedings of the 28th Annual Simulation Symposium*, p. 2-12, 1995d.

FERSCHA, A.; JOHNSON, J. A. Testbed for Parallel Simulation Performance Prediction. In: *Proceedings of the 1996 Winter Simulation Conference (WSC'96)*, 1996.

FERSCHA, A.; JOHNSON, J. Shock Resistant Time Warp. In: *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS'99)***,** p. 92-100, 1999.

FRANCÊS, C. R. L.; SANTANA, M. J.; SANTANA, R. H. C.; ORLANDI, R. C. G. S. Tools and Methodologies For Performance Evaluation of Distributed Computing Systems - A Comparison Study. In: *Proceedings of the 1997 Summer Computer Simulation Conference (SCSC'97)*, p. 13-17, 1997.

FRANCÊS, C. R. L.; SPOLON, R.; SANTANA, M. J.; SANTANA, R. H. C. *Modelagem e Especificação Utilizando Redes de Petri e Statecharts*. Nota Didática 45 do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2000.

FRANCÊS, C. R. L. *Modelagem e Avalação de Desempenho Utilizando Statecharts*. Monografia de Qualificação (Doutorado). Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2001.

FUJIMOTO, R. M. Parallel Discrete Event Simulation. *Communications of the ACM*, v. 33, nº 10, p. 31-53, 1990a.

FUJIMOTO, R. M. Performance of Time Warp under Synthetic Workloads. In: *Proceedings of the SCS Multiconference on Distributed Simulation*, p. 23-28, 1990b.

FUJIMOTO, R. M. Parallel and Distributed Discrete Event Simulation: Algorithms and Applications. In: *Proceedings of the 1993 Winter Simulation Conference (WSC'93)*, 1993.

FUJIMOTO, R. M. Parallel and Distributed Simulation. In: *Proceedings of the 1995 Winter Simulation Conference (WSC'95)*, 1995.

FUJIMOTO, R. M. *Parallel and Distributed Simulation Systems*. John Wiley & Sons, Inc., 2000.

GILL, A. *Introduction to the Theory of Finite-State Machine*. New Jork, McGraw-Hill, 1962.

GUPTA, A.; AKYILDIZ, I. F.; FUJIMOTO, R. M. Performance Analysis of Time Warp With Multiple Homogeneous Processors. *IEEE Transactions on Software Engineering* 17, nº 10, p. 1013-1027, 1991.

HAGENAUER, H. Global Virtual Time Approximation for Split Queue Time Warp. 4th International ACPC Conference. In: *Lecture Notes on Computer Science* nº 1557, p. 540-548, 1999.

HAREL, D. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming v. 8 n.3, p.231-274, 1987.

HIGGINBOTTOM, GARY N. *Performance Evaluation of Communication Networks*. Artech House, Inc., 1998.

HOEGER, H. R.; JONES, D. W. Integrating Concurrent and Conservative Distributed Discrete-Event Simulators. *Simulation*, v. 67, nº 5, p. 303-314, 1996.

HWANG, K.; XU, Z. *Scalable Parallel Computing – Technology, Architecture. Programming*. McGraw-Hill Companies, 1998.

JAIN, R. *The Art of Computer Systems Performance Analysis. Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley & Sons, Inc., 1991.

JENSEN, K. An Introduction to the Practical Use of Coloured Petri Nets. In: *Lecture Notes on Computer Science* n· 1492, p. 237-292, 1998.

JENSEN, K.; HUBER, P.; SHAPIRO, R. M. Hierarchies in Coloured Petri Nets, In: *Lectures Notes in Computer Science*, n. 483, p. 313-341, Springer-Verlag, 1990.

JEFFERSON, D. R. Virtual Time**.** *IEEE Transactions on Programming Languages and Systems*, v. 7, n. 3, p. 404-425, 1985.

JHA, V.; R. BAGRODIA. A Performance Evaluation Methodology for Parallel Simulation Protocols. In: *Proceedings of the 10th Workshop on Parallel and Distributed Simulation (PADS'96)*, p. 180-185, 1996.

JHA, V.; BAGRODIA, R. L. A Unified Framework for Conservative and Optimistic Distributed Simulation. In: *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS'94)*, p. 12-19, 1994.

JONES, K. G.; DAS, S. Parallel Execution of a Sequential Network Simulator. In: *Proceedings of the 2000 Winter Simulation Conference (WSC'2000)*, p. 418-424, 2000.

KALANTERY, N. Tentative Time Warp. In: *Proceedings of the 3th International Euro-Par Conference. In Parallel and Distributed Algorithms, Lecture Notes in Computer Science*, v. 1300, p. 458-467, 1997.

KELLY, O. E.; LAI, J.; MANDAYAM, N. B.; OGIELSKI, A. T.; PANCHAL, J.; YATES, R. D. Scalable Parallel Simulations of Wireless Networks with WiPPET: Modeling of Radio Propagation, Mobility and Protocols. *Mobile Networks and Applications*, p. 199-208, v. 5, n. 3, 2000.

KNOP, F.; MASCARENHAS, E.; REGO, V. A Parallel GPSS Based on the Parasol Simulation System. In: *Proceedings of the 1996 Winter Simulation Conference (WSC'96)*, p. 801-808, 1996.

MACDOUGALL, M. H. *Simulating Computer Systems*. MIT Press, 1987.

MACIEL, P. R. M.; LINS, R. D.; CUNHA, P. R. F. *Introdução às Redes de Petri*

e Aplicações. 10ª Escola de Computação, Instituto de Computação, Unicamp, 1996.

McGOUGH, A. S. M.; MITRANI, I. Efficient Distributed Simulation of a Communication Switch with Bursty Source and Losses. In: *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS'2000)*, p. 85-92, 2000.

MADISETTI, V. K.; HARDAKER, D. A.; FUJIMOTO, R. M. The MIMDIX Operating System for Parallel Simulation**.** In: *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS'92)*, p. 65-64, 1992.

MARSAN, M. A.; BALBO. G.; CONTE, G.. *Performance Models of Multiprocessor Systems*. MIT Series in Computer Systems. 1986.

MARSAN, M. A. Stochastic Petri Nets: An Elementary Introduction. In: *Lecture Notes in Computer Science*, n. 424, p. 1-30, 1989.

MARSAN, M. A.; BOBBIO, A.; DONATELLI, S. Petri Nets in Performance Analysis: An Introduction. In: *Lecture Notes in Computer Science*, nº 1491, p. 211-256, 1998.

MENDENHALL, W.; SINCICH, T. *Statistics for Engineering and the Sciences*. Dellen Publishing Company. 963p, 1988.

MEYER, R. A.; MARTIN, J. M.; BAGRODIA, R. L. Slow Memory: the Rising Cost of Optimism. In: *Proceedings of the 14th Workshop on Parallel and Distributed Simulation* (PADS'2000), p. 45-52, 2000.

MISRA, J. Distributed Discrete-Event Simulation**.** *ACM Computing Surveys*, v. 18, n. 1, p. 39-65, 1986.

MOORE, K. E.; HAMMER, S. ALPHA/Sim Simulation Software Tutorial. *Proceedings of the 1999 Winter Simulation Conference (WSC'99)*, p. 289-296, 1999.

MOORE, K. E.; CHIANG, JACK C. ALPHA/Sim Simulation Software Tutorial. *Proceedings of the 2000 Winter Simulation Conference (WSC'2000)*, 2000.

MORSELLI, J. C. M. JR.; SANTANA, R. H. C.; SANTANA, M. J.; ULSON, R. S. An Approach for Dynamic Swapping of Distributed Simulation Synchronisation Protocols. In: *Proceedings of European Simulation Symposium (ESS' 2000)*, 2000.

MORSELLI JR., J. C. M. *Um Mecanismo para Troca de Protocolos de Sincronização de Simulação Distribuída em Tempo de Execução.* Tese (Doutorado). Instituto de Física de São Carlos, Universidade de São Paulo, 2000.

MURATA, T. Petri Nets: Properties, Analysis and Applications. In: *Proceedings of The IEEE*, 1989.

NICOL, D. M. Performance Bounds on Parallel Self-Initiating Discrete-Event Simulations. *ACM Transactions on Modeling and Computer Simulations*, v. 1, nº 1, p. 24-50, 1991.

PASQUINI, R.; REGO, V. Optimistic Parallel Simulation over a Network of Workstations. In: *Proceedings of the 1999 Winter Simulation Conference (WSC'99)*, p. 1610-1617, 1999.

PETERSON, J. L. *Petri Nets an Introduction*, Prentice Hall, Inc., 1981.

PETRI, C. A. Kommunikation mit Automaten. Schriften des IIM Nr. 2, Institut für

Instrumentelle Mathematik, Bonn, 1962. English Translation: Technical Report RADC-TR-65-377, Griffiths Air Force Base, New York,Vol. 1, Suppl. 1, 1966.

PHAM, C. D.; FDIDA, S. Length-based Blocking and Local Estimations in Distributed Simulation: A Case Study. In: *Proceedings of the 29th Symposium on Simulation*, p.97-106, 1996.

PHAM, C. D.; BAGRODIA, R. L. Building Parallel Time-Constrained HLA Federates: A case Study with the PARSEC Parallel Simulation Language. In: *Proceedings of the 1998 Winter Simulation Conference (WSC'9*8), 1998.

PORRAS, J.; IKONEN, J.; JARMO, H. Applying a Modified Chandy-Misra Algorithm to the Distributed Simulation of a Cellular Network. In: *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98)*, p.188-195, 1998.

PRAKASH, A.; SUBRAMANIAN, R. Filter: An Algorithm for Reducing Cascaded Rollbacks in Optimistic Distributed Simulations. In: *Proceedings of the 24th Annual Simulation Symposium, Simulation Multiconference*, IEEE Press, p. 123-132, 1991.

PRAKASH, A.; SUBRAMANIAN, R. An Efficient Optimistic Distributed Simulation Scheme based on Conditional Knowledge**.** In: *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS'92),* Simulation Series, v. 24, n° 3, 1992.

QUAGLIA, F. Event History Based Sparse State Saving in Time Warp. In: *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98)*, p. 72-79, 1998.

QUAGLIA, F. Fast Software Checkpointing in Optimistic Simulation: Embedding State Saving in the Event Routine Instructions. In: *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS'99), p. 118-125, 1999.*

QUAGLIA, F.; CORTELESSA, V. Grain Sensitive Event Scheduling in Time Warp Parallel Discrete Event Simulation. In: *Proceedings of The 14th Workshop on Parallel and Distributed Simulation (PADS'2000)*, p. 173-180, 2000.

RADHAKRISHNAN, R.; MOORE, L.; WILSEY, P. A. External Adjustment of Runtime Parameters in Time Warp Synchronized Parallel Simulators. In: *Proceedings of the 11th International Parallel Processing Symposium*, p.260-266, 1997.

RADHAKRISHNAN, R.; ABU-GHAZALEH, N.; CHETLUR, M.; WILSEY, P. A. On-line Configuration on a Time Warp Parallel Discrete Event Simulator. In: *Proceedings of the International Conference on Parallel Processing (ICPP-98),* p. 28-35, 1998.

RAJAEI, H.; AYANI, R.; THORELLI, L. The Local *Time Warp* Approach to Parallel Simulations**.** In: *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS'93),* p. 119-126, 1993.

REIHER, P. L.; JEFFERSON, D. Limitation of Optimism in the Time Warp Operating System**.** In: *Proceedings of the 1989 Winter Simulation Conference (WSC'89),* p. 765-769, 1989.

REIHER, P. L.; FUJIMOTO, R.; BELLENOT, S.; JEFFERSON, David. Cancellation Strategies in Optimistic Execution Systems**.** In: *Proceedings of the*

1990 Distributed Simulation Conference, p. 112-121, 1990.

REYNOLDS Jr., P. F. Comparative Analysis of Parallel Simulation Protocols. In: *Proceedings of the 1989 Winter Simulation Conference (WSC'89)*, p. 671-679, 1989.

RÖNNGREN, R.; LILJENSTAM, M.; AYANI, R.; MONTAGNAT, J. A Comparative Study of State Saving Mechanisms for *Time Warp* Synchronized Parallel Discrete Event Simulation. In: *Proceedings of the 24th Annual Simulation Symposium (ASS'96)*, 1996a.

RÖNNGREN, R.; BARRIGA, L.; AYANI, R. An Incremental Benchmark Suite for Performance Tuning of Parallel Discrete Event Simulation. In: *Proceedings of the 29th Hawaii International Conference on System Sciences*, 1996b.

RÖNNGREN, R.; LILJENSTAM, M; AYANI, R; MONTAGNAT, J. Transparent Incremental State Saving in Time Warp Parallel Discrete Event Simulation. In: *Proceedings of the 10th Workshop on Parallel and Distributed Simulation (PADS'96)*, 1996c.

SANTANA, M. J. *An Advanced Filestore Architecture for a Multiple-LAN Distributed Computing System*. Tese (Doutorado). University of Southampton, 1989a.

SANTANA, R. H. C. *Performance Evaluation of LAN-Based File Servers*. Tese (Doutorado). University of Southampton, 1989b.

SILVA, A. R. F. *Modelos de Redes de Filas para Sistemas Computacionais Distribuídos: Simulação x Métodos Analíticos*. Dissertação (Mestrado). Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2000.

SIMMONDS, R.; BRADFORD, R.; UNGER, B. Applying Discrete Event Simulation to Network Emulation. In: *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS 2000)*, p. 15-22, 2000.

SOLIMAN, H. M. On the Selection of the State Saving Strategies in Time Warp Parallel Simulators. Transactions of the Society for Computer Simulation International, v. 16, n. 1, p. 32-36, 1999.

SPOLON, R; SANTANA, M. J.; SANTANA, R. H. C. Distributed Simulation, Time Warp and Its Variants: Taxonomy and Performance Evaluation Issues. In: *Proceedings of the 13th European Simulation Multiconference (ESM'99)*, p. 220-227, 1999.

SPOLON, R.; SANTANA, M. J.; SANTANA, R. H. C. A Methodology for Performance Evaluation of Optimistic Distributed Simulation Synchronisation Mechanisms. In: *Proceedings of the 14th European Simulation Multiconference (ESM'2000)*, p. 121-125, 2000a.

SPOLON, R.; ULSON, R. S.; SANTANA, M. J.; SANTANA, R. H. C. *Simulação Distribuída, Time Warp e Variantes: Descrição e Análise Comparativa*. Relatórios Técnicos do ICMC-USP, n. 111, Instituto de Ciências Matemáticas de São Carlos da Universidade de São Paulo, 2000b.

SRINIVASAN, S.; REYNOLDS Jr.; P. F. NPSI Adaptive Synchronization Algorithms for PDES. In: *Proceedings of the 1995 Winter Simulation Conference (WSC'95)*, p. 658-665, 1995a.

SRINIVASAN, S.; REYNOLDS Jr.; P. F. Adaptive Algorithms vs. Time Warp:

An Analytical Comparison. In: *Proceedings of the 1995 Winter Simulation Conference (WSC'95)*, p. 666-673, 1995b.

STEINMAN, J. S. SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation. In: *Proceedings of the SCS Multiconference on Advances on Parallel and Distributed Simulation*, p. 95-103, 1991.

STEINMAN, J. S. SPEEDES: A Unified Approach to Parallel Simulation. In: *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS'92)*, p. 75-84, 1992.

STEINMAN, J. S. Breathing Time Warp. In: *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS'93)*, p. 109-118, 1993.

STEINMAN, J. 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).*

STEINMAN, J. S.; LEE, C. A.; WILSON, L. F.; NICOL, D. M. Global Virtual Time and Distributed Synchronization. In: *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS'95)*, p. 139-148, 1995.

TAY, S. C.; TEO, Y. M.; KONG, S. T. Speculative Parallel Simulation with an Adaptive Throttle Scheme. In: *Proceedings of the 11th Workshop on Parallel and Distributed Simulation (PADS'97)*, p. 116-123, 1997.

TAY, S. C.; TEO, Y. M.; AYANI, R. Performance Analysis of Time Warp Simulation with Cascading Rollbacks. In: *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98)*, p. 30-37, 1998.

TURNER, S. J.; XU, M. Q. Performance Evaluation of the Bounded Time Warp Algorithm. In: *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS'92)*, p. 117-126, 1992a.

TURNER, S. J.; XU, M. Q. A Portable Parallel Discrete Event Simulation System. In: *Proceedings of ESPRIT Workshop on Parallel Computing*, p. 284-287, 1992b.

ULSON, R. S.; SANTANA, R. H. C.; SANTANA, M. J. Distributed Simulation Environment for Computing Systems Performance Evaluation. In: *Proceedings of the 1997 Summer Computer Simulation Conference – Symposium of Performance Evaluation of Computer and Telecommunication Systems (SPECTS 97)*, p. 85-90, The Society for Computer Simulation International, July 1997.

ULSON, R. S. *Simulação Distribuída em Plataformas de Portabilidade: Viabilidade de Uso e Comportamento do Protocolo CMB.* Tese (Doutorado). Instituto de Física de São Carlos, Universidade de São Paulo, 1999a.

ULSON, R. S.; MORSELLI Jr., J. C. M.; SANTANA, R. H. C.; SANTANA, M. J. Conservative Distributed Simulation on Portability Platforms: The CMB Protocol Behaviour. In: *Proceedings of the IASTED International Conference on Applied Modelling and Simulation (AMS'99)*, Cairns, Queensland, Austrália, 1999b.

UNGER, B.; ZIAO, X.; CLEARY, J.; TSAI, J.; WILLIAMSON, C. Parallel Shared-Memory Simulator Performance for Large ATM Networks. *ACM Transactions on Modeling and Computer Simulation*, p. 358-391, v. 10, n. 4,

2000.

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

YOUNG, C. H.; WILSEY, P. A. Optimistic Fossil Collection for Time Warp Simulation. In: *Proceedings of the Hawaiian International Conference on System Sciences (HICSS'96)*, p. 364-372, 1996.

YOUNG, C. H.; ABU-GHAZALEH, N. B.; WILSEY, P. A. OFC: A Distributed Fossil Collection Algorithm for Time Warp. In: *Proceedings of the 12th International Conference on Distributed Computing (DISC'98)*, 1998.

YOUNG, C. H.; RADHAKRISHNAN, R.; WILSEY, P. A. In: *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS'99)*, p. 136, 1999.

ZHANG, Y.; CAI, W.; TURNER, S. J. Parallel Discrete Event Simulation of Manufacturing Systems Using Parsec. In: *Proceedings of the 14th European Simulation Multiconference (ESM'2000)*, p. 296-301, 2000.

Apêndice A

Ferramenta Utilizada para Representação e Simulação do Modelo

O *ALPHA/Sim* é uma ferramenta de simulação de propósito geral e de domínio privado, que utiliza as características das Redes de Petri Estocásticas Generalizadas, das Redes de Petri Coloridas e das Redes de Petri Hierárquicas, como técnica para efetuar a modelagem. Através de um editor gráfico o *ALPHA/Sim* possibilita ao usuário desenhar e fornecer todos os parâmetros necessários ao modelo de simulação (Moore; Chiang 2000). Após a definição e parametrização do modelo, o usuário pode executar a simulação sem a necessidade de escrever o código correspondente. O *ALPHA/Sim* também coleta, automaticamente, as estatísticas da simulação, como por exemplo as relacionadas com filas e atrasos, e faz verificação das expressões aritméticas e da existência de laços infinitos (Alphatech Corporation 1995). A figura A.1 apresenta a tela inicial da ferramenta.

Figura A.1: Tela Inicial do *ALPHA/Sim*.

Os ícones da barra lateral permitem ao usuário desenhar a Rede de Petri. A figura A.2 apresenta um modelo simples em Redes de Petri, no qual clientes chegam a um sistema (transição *Gera_Clientes*), entram na fila (lugar *Fila*) para solicitar atendimento ao único servidor (transição *Servidor*) e depois vão embora (lugar *Serviço_Completo*). A transição *Sorvedouro* elimina os *tokens* que não são mais necessários à Rede de Petri.

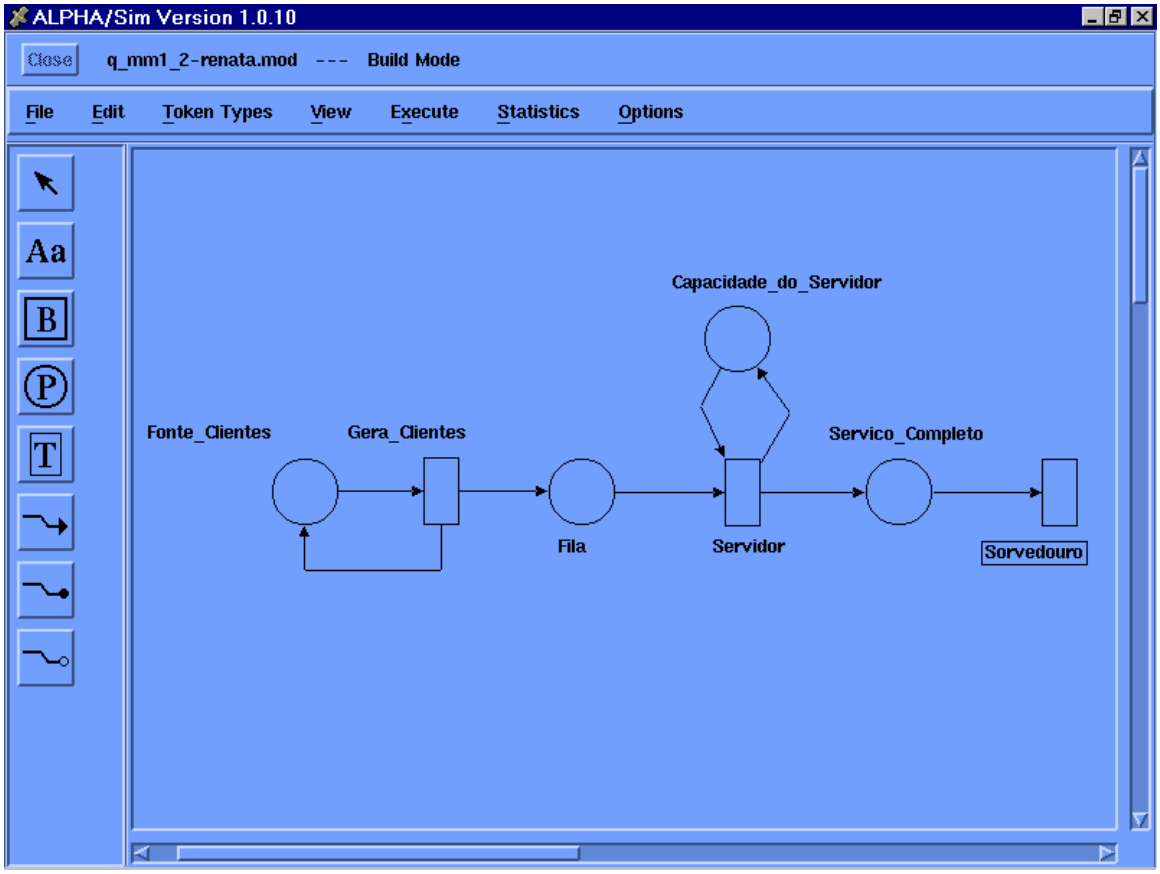


Figura A.2: Utilizando *ALPHA/Sim* para Descrever um Modelo Simples.

A definição do tempo entre-chegadas de novos clientes ao sistema ocorre na transição *Gera_Clientes*, como mostrado na figura A.3. Uma janela de parametrização permite ao usuário escolher entre uma transição imediata ou uma transição temporizada, com o uso de diferentes distribuições de probabilidade ou regras definidas pelo usuário. Nesse exemplo, a distribuição utilizada é a exponencial, com média 10,0 unidades de tempo (opção *Transition Timing Rule* da janela de parametrização da transição). A parametrização do tempo de serviço é mostrada na figura B.4, onde foi escolhido o tempo de serviço constante igual a 8,5 unidades de tempo. Nas figuras A.3 e A.4 estão apresentadas as opções para a obtenção de estatísticas, na lista de opções *Output Statistics*.

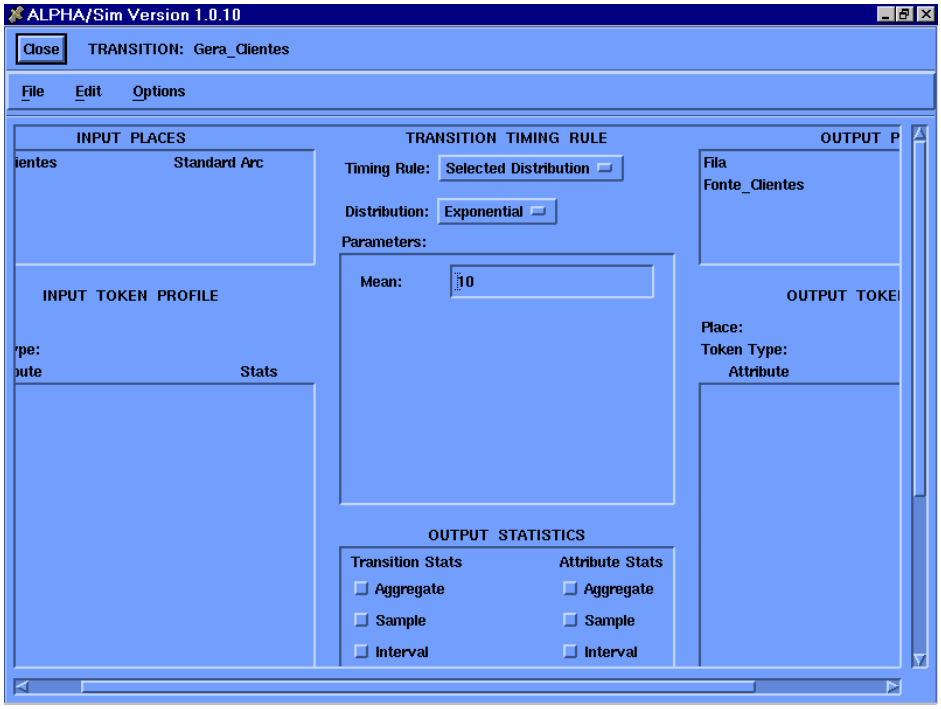


Figura A.3: Parametrização da Transição *Gera_Clientes*.

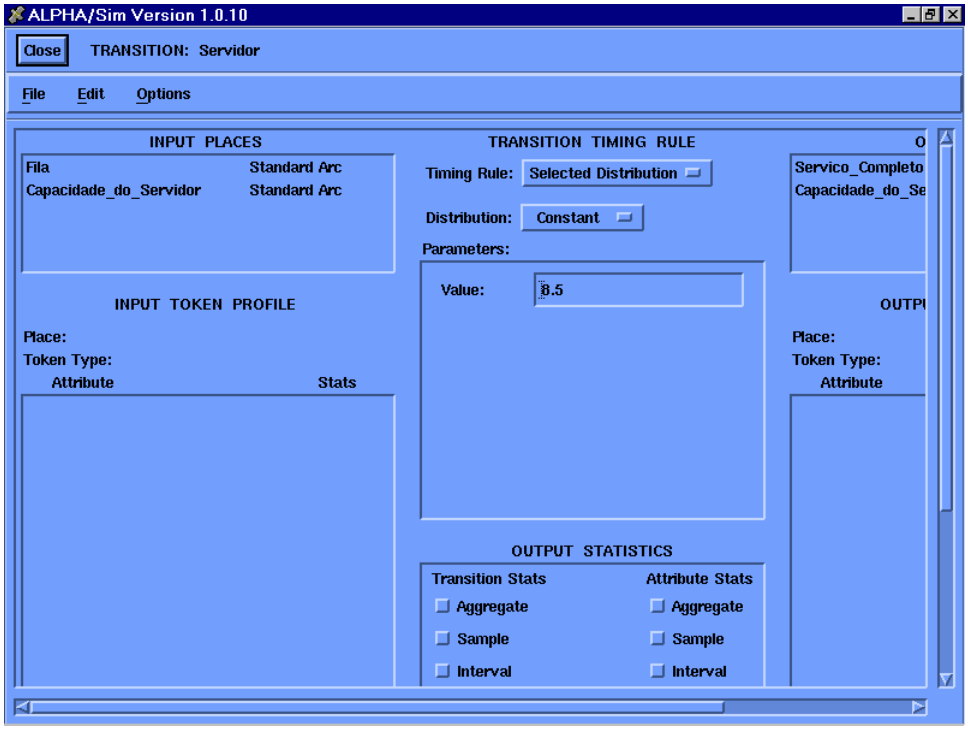


Figura A.4: Parametrização da Transição *Servidor*.

A figura A.5 mostra a janela de parametrização do lugar *Capacidade do Servidor*, iniciado com o valor 1 (campo *Initial Tokens*). A cor do *token*, isto é, o seu tipo, é escolhido através do campo *Token Id*. Os atributos de um *token* são construídos através do menu da barra superior da janela principal, escolhendo-se a opção *Token Types* (figura A.6). Pode-se criar um novo tipo, alterar ou apagar um tipo de *token* (figura A.7).

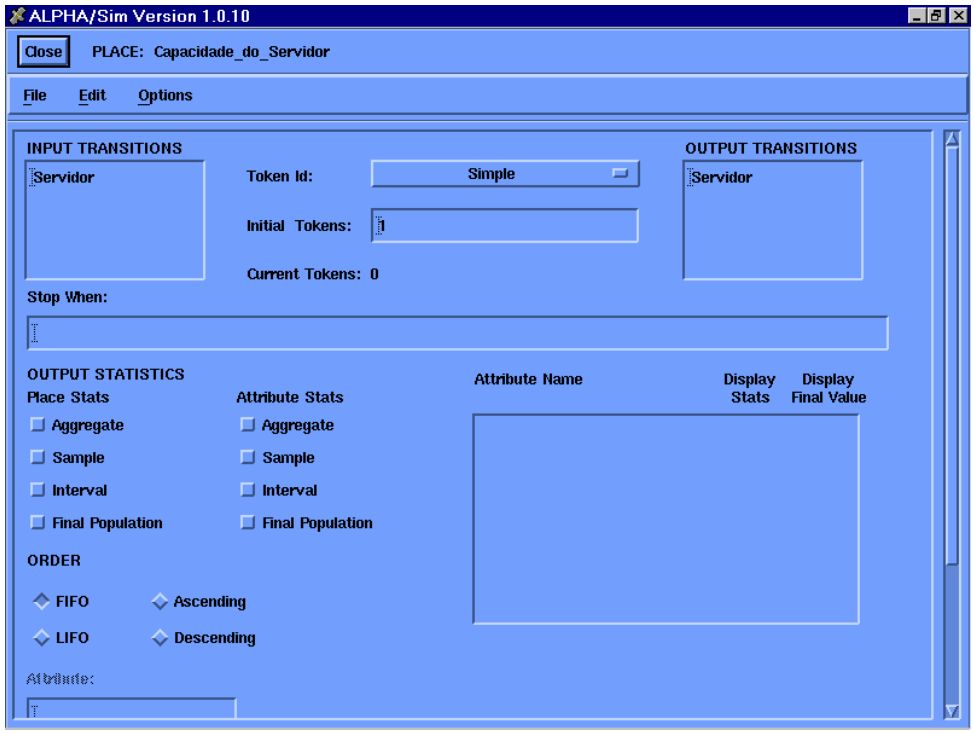


Figura A.5: Parametrização do Lugar *Capacidade do Servidor*.

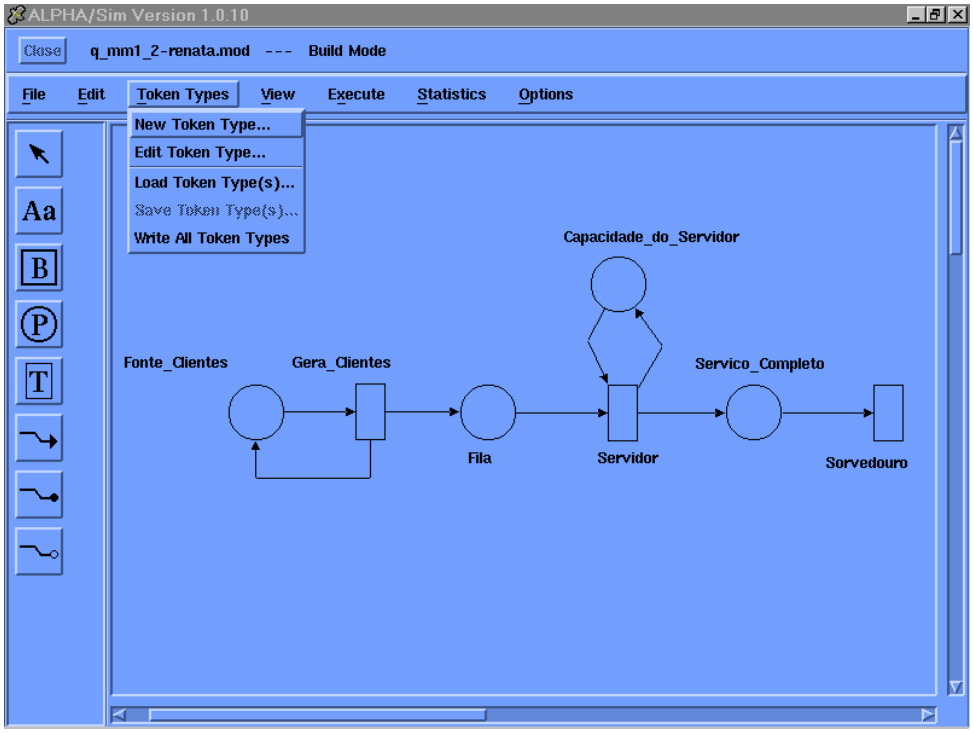


Figura A.6: Opção do Menu para Manipulação de *Tokens*.

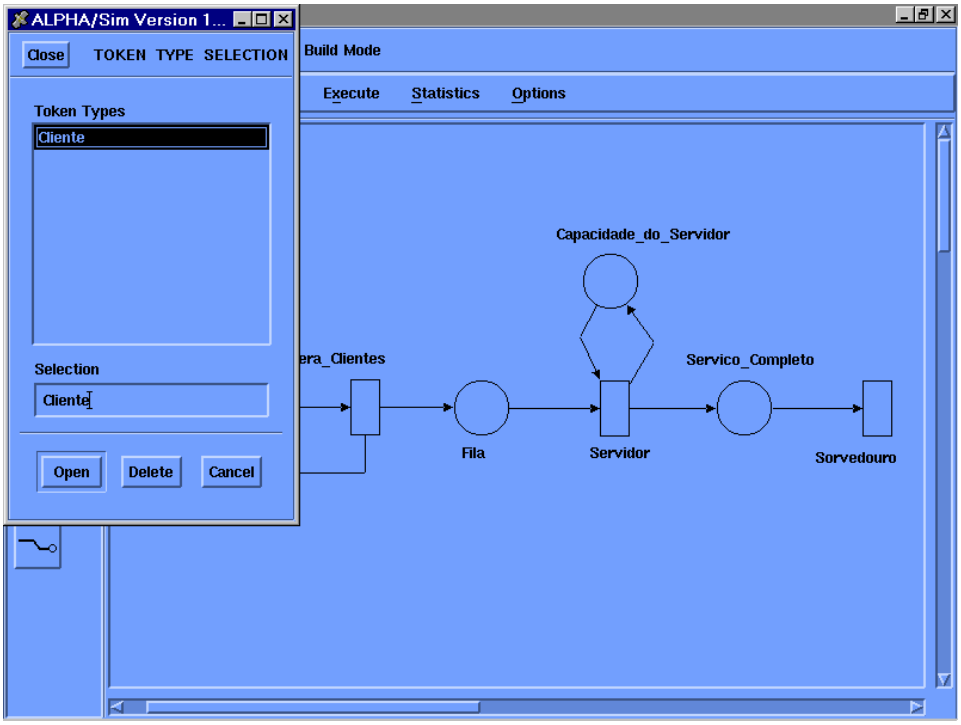


Figura A.7: Abrindo a Especificação de um Tipo de *Token*.

A figura A.8 apresenta a janela utilizada para criar os atributos de um tipo de *token*. O tipo de *token Cliente* (campo *Token Type Name*) possui dois atributos (campos *Attribute Name*): *Type*, do tipo Inteiro, e *Time*, do tipo Real.

Figura A.8: Janela para Definir Atributos de um Tipo de *Token*.

A atribuição de valores aos atributos de um *token* ocorre, inicialmente, na sua criação (se necessário) e, posteriormente, durante a execução da simulação, através das transições (figura A.9). O atributo é selecionado (campo *Attribute*) e na opção *Definition* de *Construction of Output Attributes* o usuário pode definir o valor de saída. Essa definição pode utilizar valores numéricos ou expressões aritméticas.

Antes da execução da simulação, ou a qualquer momento durante a construção do modelo, o *ALPHA/Sim* permite que a consistência da especificação em Redes de Petri seja verificada, através da opção *Check Expressions* do Menu *Execute* (figura A.10). Após a execução total ou parcial da simulação pode-se obter as estatísticas referentes a lugares e transições, como mostra a figura A.11.

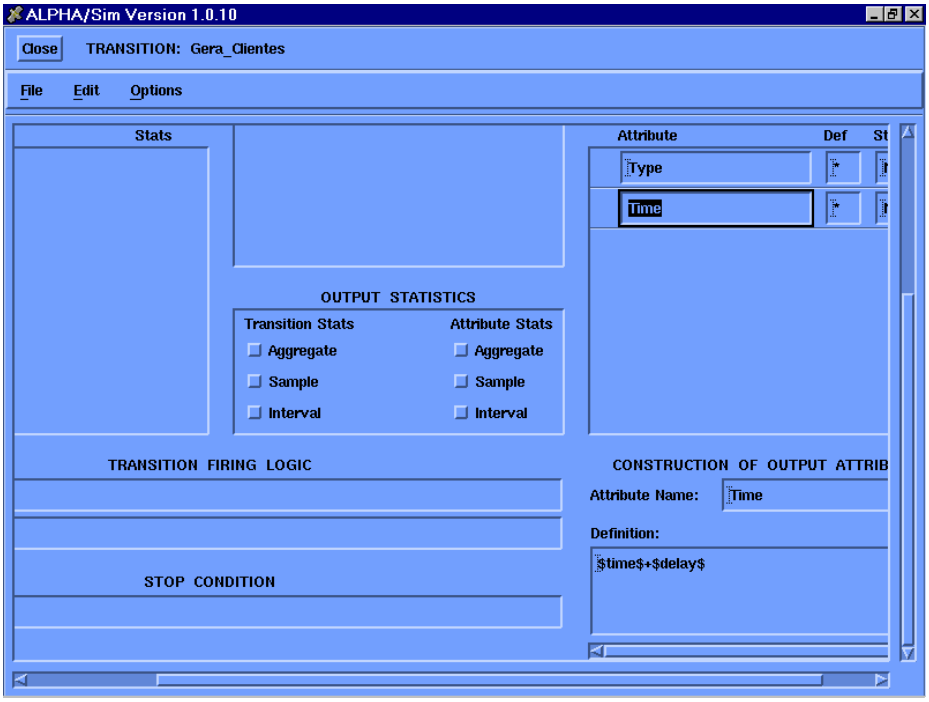


Figura A.9: Definição do Valor de Saída do Atributo *Time*.

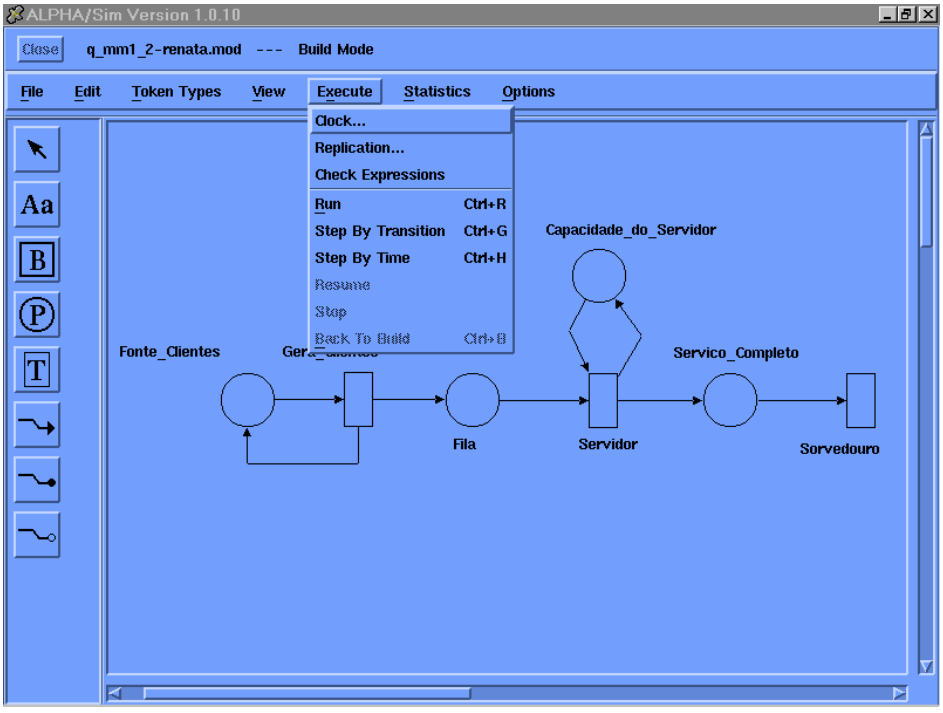


Figura A.10: Menu *Execute*.

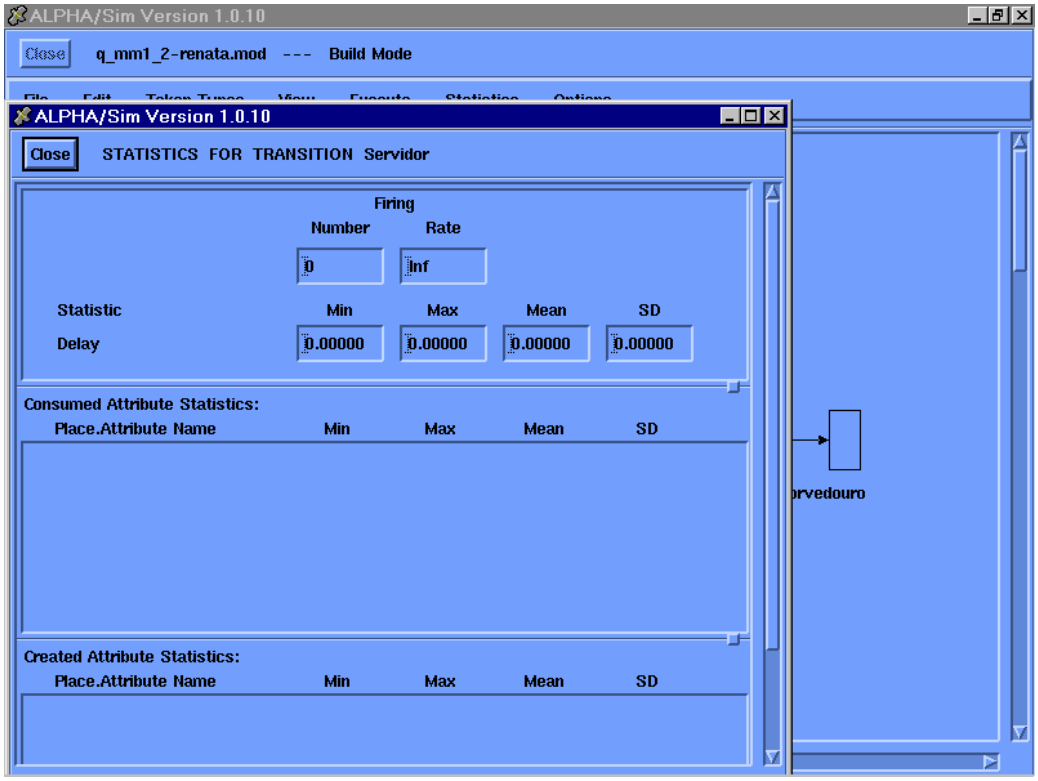


Figura A.11: Estatísticas Referentes à Transição *Servidor*.

O *ALPHA/Sim* está disponível em uma versão acadêmica para microcomputadores executando o sistema operacional Windows NT 4.0 ou para estações de trabalho Sun executando o sistema operacional Unix (Solaris e SunOS). A versão para Windows NT foi adquirida pelo Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP e utilizada no desenvolvimento, verificação e validação do modelo do *Time Warp*, porém apresentou problemas de implementação que não permitiam o volume necessário de execuções da simulação para a obtenção de resultados precisos. Esses problemas só foram observados devido à complexidade do modelo proposto. Assim, passou-se a utilizar a versão do ambiente *ALPHA/Sim* para UNIX, liberada exclusivamente para o desenvolvimento desta tese. A versão UNIX executa normalmente o modelo do *Time Warp* proposto sem apresentar os problemas encontrados na versão para Windows NT.

Apêndice B

Algoritmo dos Blocos

B.1 Descrição das Estruturas de Dados

B.1.1 *Input_Queue*

Armazena as informações referentes às mensagens que aguardam para serem executadas; tem a mesma estrutura em todos os Processos Lógicos. Os *tokens* são ordenados pelo *timestamp*. Contém inicialmente um *token*, que representa a chegada do primeiro cliente na aplicação.

```
PLACE: Input_Queue_P1

Token Type in Place: EventMessage

Number of Initial Tokens: 1
Stop When: (null)
Place Statistics:
Aggregate:  N
Sample:     N
Interval:   N
Final Population:   N

Attribute Statistics:
Aggregate:  N
Sample:     N
Interval:   N
Final Population:   N
```

```
Order:ASCENDING
Order Attribute: Timestamp
Branching Rule: Constructed
Condition            Destination

Initial Tokens:

Initial Token: 1
Attribute Name       Statistics   Initial Value
Signal                 N            True
Timestamp              N            0
LVT                    N            0
NewEvent               N            True
TimestampAntQueue      N            0
CountAntQueue          N            0
CountAux               N            0
LVTLinha               N            0

Attribute Display Flags:

Attribute Name       Display     Display
                      Stats      Final Value
Signal                 N            N
Timestamp              N            N
LVT                    N            N
NewEvent               N            N
TimestampAntQueue      N            N
CountAntQueue          N            N
CountAux               N            N
LVTLinha               N            N

Current Tokens:

Final Place Population: 0
```

B.1.2 *Input_Queue_Exec*

Armazena as informações referentes aos eventos que já foram executados. Os *tokens* são organizados de forma *FIFO* (*First-In, First-Out*).

```
PLACE: Input_Queue_Exec_P1

Token Type in Place: QueueMessage

Number of Initial Tokens: 0
Stop When: (null)
Place Statistics:
Aggregate:  N
Sample:     N
Interval:   N
Final Population:   N

Attribute Statistics:
Aggregate:  N
Sample:     N
```

```
Interval:   N
Final Population:   N

Order:FIFO
Order Attribute:
Branching Rule: Probability
Condition            Destination
0.2                          T_Inc_Search_Input_Queue_Exec_P1
0.2                          T_Inc_Search_Wrong_Exec_P1
0.2                          T_Inc_Search_Wrong_Exec_P1b
0.2                          T_Busca_LVTLinhaP_1
0.2                          T_Busca_LVTLinhaP_1b

Initial Tokens:

Attribute Display Flags:

Attribute Name       Display     Display
                      Stats      Final Value
Timestamp              N           N
NewEvent               N           N
LVTLinha               N           N

Current Tokens:

Final Place Population: 0
```

B.1.3 *Output_Queue*

Armazena as mensagens que o Processo Lógico envia para execução remota. Ordenada por *LIFO* (*Last-In, First-Out*).

```
PLACE: Output_Queue_P1

Token Type in Place: QueueMessageOutput

Number of Initial Tokens: 0
Stop When: (null)
Place Statistics:
Aggregate:  N
Sample:     N
Interval:   N
Final Population:   N

Attribute Statistics:
Aggregate:  N
Sample:     N
Interval:   N
Final Population:   N

Order:LIFO
Order Attribute:
Branching Rule: Priority
Condition            Destination
0.5                          T_Inc_Search_OutputQueue_P1
```

```
0.5                          T_Inc_Search_OutputQueue_P1b

Initial Tokens:

Attribute Display Flags:

Attribute Name       Display     Display
                      Stats      Final Value
TimestampOutput        N            N
TimestampMsgGerouOutputN            N

Current Tokens:

Final Place Population: 0
```

B.1.4 *Ant_Queue*

Armazena as antimensagens que o Processo Lógico recebe, e que chegam antes da mensagem correspondente.

```
PLACE: AntQueue_P1

Token Type in Place: TokenAntQueue

Number of Initial Tokens: 0
Stop When: (null)
Place Statistics:
Aggregate:  N
Sample:     N
Interval:   N
Final Population:   N

Attribute Statistics:
Aggregate:  N
Sample:     N
Interval:   N
Final Population:   N

Order:FIFO
Order Attribute:
Branching Rule: Probability
Condition            Destination
0.5                          T_Inc_Search_AntQueue_P1
0.5                          T_Inc_Search_AntQueue_P1b

Initial Tokens:

Attribute Display Flags:

Attribute Name       Display     Display
                      Stats      Final Value
TimestampAntQueue      N            N

Current Tokens:
Final Place Population: 0
```

B.2 Tipos de *Tokens* (atributos) Utilizados

```
ALPHA/Sim Version 1.0.10
Fri Jun 19 11:01:39 1998
Model Name 'twsimulation4.mod'

Token Type Definition of EventMessage

Attribute Name            Class            Type             Initial
Range

Signal                    Scalar           Boolean
Timestamp                 Scalar           Real
LVT                       Scalar           Real
NewEvent                  Scalar           Boolean
TimestampAntQueue         Scalar           Real
CountAntQueue             Scalar           Integer
CountAux                  Scalar           Integer
LVTLinha                  Scalar           Real

Token Type Definition of EventMessageSearch

Attribute Name            Class            Type             Initial
Range

Signal                    Scalar           Boolean
Timestamp                 Scalar           Real
LVT                       Scalar           Real
NewEvent                  Scalar           Boolean
TimestampAntQueue         Scalar           Real
LVTLinha                  Scalar           Real
Continue                  Scalar           Boolean
CountAntQueue             Scalar           Integer
CountAux                  Scalar           Integer
LVTLinhaAntQueue          Scalar           Real

Token Type Definition of EventMessageSearchOutput

Attribute Name            Class            Type             Initial
Range

Signal                    Scalar           Boolean
Timestamp                 Scalar           Real
LVT                       Scalar           Real
NewEvent                  Scalar           Boolean
TimestampAntQueue         Scalar           Real
CountAntQueue             Scalar           Integer
CountAux                  Scalar           Integer
LVTLinha                  Scalar           Real
TimestampMsgGerouOutput   Scalar           Real
```

Utilizado quando é necessário efetuar busca na *Output_Queue*, para enviar as antimensagens em caso de *rollback*.

```
Token Type Definition of QueueMessage
```

Attribute Name	Class	Type	Initial Range
Timestamp	Scalar	Real	
NewEvent	Scalar	Boolean	
LVTLinha	Scalar	Real	

Token Type Definition of **QueueMessageOutput**

Attribute Name	Class	Type	Initial Range
TimestampOutput	Scalar	Real	
TimestampMsgGerouOutput	Scalar	Real	

Utilizado para armazenar as mensagens na Output_Queue, enviadas para execução remota.

Token Type Definition of **TokenAntQueue**

Attribute Name	Class	Type	Initial Range
TimestampAntQueue	Scalar	Real	

Utilizado para armazenar as informações da antimensagem que chega antes da mensagem correspondente.

Token Type Definition of **Token_Ready**

(Utilizado no Lugar Ready, guarda o LVT e o LVTLinha do LP)

Attribute Name	Class	Type	Initial Range
LVT	Scalar	Real	
LVTLinha	Scalar	Real	

Armazena o *LVT* e o *LVTLinha* do Processo Lógico.

B.3 Algoritmos dos Processos Lógicos

Um Processo Lógico efetua basicamente duas atividades, que envolvem a execução de mensagens de evento e a recepção das mensagens enviadas por Processos Lógicos remotos. As seções a seguir detalham essas atividades e os algoritmos envolvidos.

Sempre que o *Buffer* de Recepção possui alguma mensagem/antimensagem, o Processo Lógico efetua o tratamento necessário.

B.3.1 Execução

B.3.1.1 Verificar Antimensagem

Uma mensagem de evento armazenada na *Input_Queue* não pode ser executada se a antimensagem correspondente chegou antes e está armazenada na *Ant_Queue*. A verificação se a antimensagem é correspondente a uma mensagem que ainda não foi executada também é feita aqui.

```
Se Ant_Queue vazia então
  Execução_Local
Senão
  CountAux = 1
  CountAntQueue = tamanho da Ant_Queue
  Enquanto CountAux <= CountAntQueue faça
    CountAux = CountAux + 1
    TimestampAntQueue = Ant_Queue.TimestampAntQueue
    Se Timestamp == TimestampAntQueue então
      CountAux = CountAntQueue + 1
    Senão
      Ant_Queue.TimestampAntQueue = TimestampAntQueue
    Fim Se
  Fim Enquanto
  Se Timestamp = TimestampAntQueue então
    Ready.LVT = LVT
    Ready.LVTLinha = LVTLinha
    Retorna para Ready  // A antmsg correspondente
  Senão                 // chegou antes ou chegou e a
                        // mensagem não tinha sido executada
                        // Ambas são descartas
    Execução_Local      // A msg pode ser executada
  Fim Se
Fim Se
```

B.3.1.2 Execução Local

Calcula o novo *LVT* do Processo Lógico e, se necessário, a marca de tempo da mensagem que vai ser enviada.

A probabilidade p de um cliente deixar o sistema, ou entrar na fila de outro recurso, que é um parâmetro relacionado à aplicação, é utilizada como probabilidade na tomada de decisão para o caminho a ser seguido pelo *token* que percorre a Rede de Petri.

```
Com probabilidade p  //O cliente deixa o sistema,
                     // o Processo Lógico não precisa enviar
                     // mensagem para execução remota
      LVT = Timestamp
      Se Timestamp < LVTLinha então  // Atualiza o termino
            LVTLinha = LVTLinha + exponencial(10.0)
      Senão
```

```
            LVTLinha = Timestamp + exponencial(10.0)
      Fim Se
      // Armazena a mensagem executada na Input_Queue_Exec
      Input_Queue_Exec.Timestamp = Timestamp
      Input_Queue_Exec.NewEvent = NewEvent
      Input_Queue_Exec.LVTLinha = LVTLinha
      Calcular nova chegada

Com probabilidade 1 –p  // O cliente não deixa o sistema, o
                        // Processo Lógico envia mensagem para
                        // execução remota
      LVT = Timestamp
      Se Timestamp < LVTLinha então  // Atualiza o termino
            Timestamp = LVTLinha + exponencial(10.0)
      Senão
            Timestamp = Timestamp + exponencial(10.0)
      Fim Se
      LVTLinha = Timestamp
      // Armazena a mensagem executada na Input_Queue_Exec
      Input_Queue_Exec.Timestamp = Timestamp
      Input_Queue_Exec.NewEvent = NewEvent
      Input_Queue_Exec.LVTLinha = LVTLinha
      Enviar mensagem para execução remota

      // Alteração inserida no LP1, 3 LPs
      // P1 decide para qual LP enviar, guarda essa informação
      // no campo Rec
```

B.3.1.3 Enviar Mensagem para Execução Remota

A mensagem é submetida à transição *T_Transmission*, para ser entregue ao Processo Lógico destino. O atributo *Signal* é marcado como *True*, significando que uma mensagem vai ser enviada.

```
Output_Queue.TimestampOutput = Timestamp
Output_Queue.TimestampMsgGerouOutput = LVT
Signal = True
Calcular nova chegada e submeter para transmissão
```

B.3.1.4 Calcular Nova Chegada

Se o campo *NewEvent* é igual a *True*, significa que uma nova chegada, na aplicação, deve acontecer.

Esse módulo somente deve estar presente no Processo Lógico que recebe entradas de clientes, ou seja, que executa as chegadas de clientes da aplicação.

```
Se NewEvent == True então
  CountAux = número de chegadas NUMDEF deve ser setado no
   Alpha/Sim
  Se Numero de Chegadas < NUMDEF então
    Timestamp = LVT + exponencial(5.0)
    NewEvent = True
```

```
    Signal = True
      // A mensagem é inserida no Buffer de recepção do
      // Processo Lógico
  Fim Se
Fim Se
Ready.LVT = LVT
Ready.LVTLinha = LVTLinha
Retorna para Ready
```

B.3.2 Recepção

A recepção das mensagens e antimensagens cuida para que as mensagens que chegam com marca de tempo menor do que o relógio local do Processo Lógico e as antimensagens sejam tratadas adequadamente. Isto é, um mecanismo de *rollback* é efetuado para deixar o Processo em um estado consistente.

As mensagens e antimensagens que chegam da rede de comunicação são armazenadas no *Buffer* de Recepção do Processo Lógico e tratadas oportunamente.

B.3.2.1 Recepção do Meio Físico

As mensagens e antimensagens que chegam pela rede de comunicação são armazenadas no Input*_Buffer*, ordenado por *FIFO* (*First*-In, *First*-*Out*).

```
Input_Buffer.Timestamp = Timestamp
Input_Buffer.Signal = Signal
Input_Buffer.NewEvent = NewEvent
```

B.3.2.2 Tratamento da Mensagem Recebida

O sinal da mensagem é analisado e o tratamento adequado é efetuado.

```
Se Signal = True então // É mensagem
      Tratar Mensagem
Senão
      Tratar Antimesagem
Fim Se

Tratar Mensagem
      Se Ant_Queue está vazia  // A antimensagem correspodente
                               // não chegou
            Se Timestamp >= LVT // Recebe a mensagem
                  Input_Queue.Timestamp = Timestamp
                  Input_Queue.Signal = Signal
                  Input_Queue.NewEvent = NewEvent
                  Ready.LVT = LVT
                  Ready.LVTLinha = LVTLinha
                  Retorna para Ready
            Senão
                  Rollback Causado Por Straggler
```

```
            Fim Se
        Senão // Efetuar busca na Ant_Queue
            CountAntQueue = Tamanho da Ant_Queue
            CountAux = 1
            Enquanto CountAux <= CountAntQueue faça
                CountAux = CountAux + 1
                                    TimestampAntQueue =
                Ant_Queue.TimestampAntQueue
                Se Timestamp == TimestampAntQueue então
                    CountAux = CountAntQueue + 1
                Senão
                    Ant_Queue.TimestampAntQueue=
                    TimestampAntQueue
            Fim Enquanto
            Se Timestamp = TimestampAntQueue então
            // A Antmsg chegou, ambas são descartas
                Ready.LVT = LVT
                Ready.LVTLinha = LVTLinha
                Retorna para Ready
            Senão
            // A Antmsg não chegou, verificar Timestamp
                Se Timestamp >= LVT // Recebe a mensagem
                    Input_Queue.Timestamp = Timestamp
                    Input_Queue.Signal = Signal
                    Input_Queue.NewEvent = NewEvent
                    Ready.LVT = LVT
                    Ready.LVTLinha = LVTLinha
                    Retorna para Ready
                Senão
                    Rollback Causado Por Straggler
                Fim Se
            Fim Se
        Fim Se

Tratar Antimensagem
  Se Input_Queue_Exec está vazia então  // A msg não chegou
    Ant_Queue.TimestampAntQueue = Timestamp
    Ready.LVT = LVT
    Ready.LVTLinha = LVTLinha
    Retorna para Ready
  Senão   // Verificar se msg correspondente já foi executada
    CountAntQueue = Tamanho da Input_Queue_Exec
    CountAux = 1
    Continue = True
    Enquanto CountAux <= CountAntQueue faça
      TimestampAntQueue = Input_Queue_Exec.Timestamp
      LVTLinhaAntQue = Input_Queue_Exec.LVTLinha
      Se Timestamp = TimestampAntQueue então
        Continue = False
        CountAux = CountAux + 1
      Senão
        Input_Queue_Exec.Timestamp = TimestampAntQueue
        Input_Queue_Exec.LVTLinha = LVTLinhaAntQueue
        CountAux = CountAux + 1
      Fim Se
    Fim Enquanto
    Se Continue == False // Achou a msg correspondente

    Rollback Causado por Antimensagem
    Senão  // A antmsg é inserida na Ant_Queue
```

```
      Ant_Queue.TimestampAntQueue = Timestamp
      Ready.LVT = LVT
      Ready.LVTLinha = LVTLinha
    Fim Se
  Fim Se

Rollback Causado Por Straggler
  // Inserir straggler na Input_Queue
  Input_Queue.Timestamp = Timestamp
  Input_Queue.Signal = Signal
  Input_Queue.NewEvent = NewEvent
  // Atualizar LVT
  LVT = Timestamp
  CountAntQueue = tamanho da Output_Queue
  CountAux = 1
  Enquanto (CountAux <= CountAntQueue) && (Output_Queue não
   vazia) faça
    TimestampAntQueue = Output_Queue.TimestampOutput
    TimestampMsgGerouOutput =
      Output_Queue.TimestampMsgGerouOutput
    CountAux = CountAux + 1
    Se TimestampMsgGerouOutput > Timestamp então
      //Enviar a antimensagem correspondente
      Transmission.Signal = False
      Transmission.Timestamp = TimestampAntQueue
      Transmission.NewEvent = False
    Senão  // Devolve a msg para a Output_Queue e encerra
           // a busca
      CountAux = CountAntQueue + 1
      Output_Queue.TimestampOutput = TimestampAntQueue
      Output_Queue.TimestampMsgGerouOutput=
      TimestampMsgGerouOutput
    Fim Se
  Fim Enquanto
  // Procurar todas as msgs executadas erroneamente
  CountAntQueue = tamanho da Input_Queue_Exec
  CountAux = 1
  Continue = False
  Enquanto (CountAux <= CountAntQueue) e (Input_Queue_Exec não
   vazia) faça
    CountAux = CountAux + 1
    LVTLinhaAntQueue = Input_Queue_Exec.LVTLinha
    TimestampAntQueue = Input_Queue_Exec.Timestamp
    Se   TimestampAntQueue   >   Timestamp   //   Voltar   para
     Input_Queue
      Input_Queue.Timestamp = TimestampAntQueue
      Input_Queue.Signal = Signal
      Input_Queue.NewEvent = False
    Senão
      Input_Queue_Exec.Timestamp = TimestampAntQueue
      Input_Queue_Exec.LVTLinha = LVTLinhaAntQueue
      Input_Queue_Exec.NewEvent = NewEvent
    Fim Se
  Fim Enquanto
  // Restaurar o LVTLinha, deve ser o da última msg executada
  // Essa msg é a última da Input_Queue_Exec
  CountAntQueue = tamanho da Input_Queue_Exec
  CountAux = 1
  Enquanto CountAux <= CountAntQueue faça
    TimestampAntQueue = Input_Queue_Exec.Timestamp
```

```
    LVTLinha = Input_Queue_Exec.LVTLinha
    CountAux = CountAux + 1
    Input_Queue_Exec.Timestamp = TimestampAntQueue
    Input_Queue_Exec.LVTLinha = LVTLinha
  Fim Enquanto
  // Retonar para Ready, com LVT e LVTLinha corretos
  Ready.LVT = LVT
  Ready.LVTLinha = LVTLinha
  Retornar para Ready
```

Rollback Causado por Antimensagem

```
CountAux = 1
CountAntQueue = tamanho da Output_Queue
  Enquanto (CountAux <= CountAntQueue) && (Output_Queue não
   vazia) faça
    TimestampAntQueue = Output_Queue.TimestampOutput
    TimestampMsgGerouOutput=
     Output_Queue.TimestampMsgGerouOutput
    CountAux = CountAux + 1
    Se TimestampMsgGerouOutput >= Timestamp então
      // Enviar a antimensagem correspondente
      Transmission.Signal = False
      Transmission.Timestamp = TimestampAntQueue
      Transmission.NewEvent = False
    Senão  // Devolve a msg para a Output_Queue e
           // encerra a busca
      CountAux = CountAntQueue + 1
      Output_Queue.TimestampOutput = TimestampAntQueue
      Output_Queue.TimestampMsgGerouOutput=
      TimestampMsgGerouOutput
    Fim Se
  Fim Enquanto
// Procurar todas as msgs executadas erroneamente
  CountAntQueue = tamanho da Input_Queue_Exec
  CountAux = 1
  Continue = False
  Enquanto (CountAux <= CountAntQueue) e (Input_Queue_Exec não
   vazia) faça
    CountAux = CountAux + 1
    LVTLinhaAntQueue = Input_Queue_Exec.LVTLinha
    TimestampAntQueue = Input_Queue_Exec.Timestamp
    Se  TimestampAntQueue  >  Timestamp  //  Voltar  para
     Input_Queue
      Input_Queue.Timestamp = TimestampAntQueue
      Input_Queue.Signal = Signal
      Input_Queue.NewEvent = False
    Senão  // Retorna a Input_Queue_Exec
      Input_Queue_Exec.Timestamp = TimestampAntQueue
      Input_Queue_Exec.LVTLinha = LVTLinhaAntQueue
      Input_Queue_Exec.NewEvent = NewEvent
    Fim Se
  Fim Enquanto
// Restaurar o LVT e LVTLinha, deve ser o da última
// msg executada
// Essa msg é a última da Input_Queue_Exec
CountAntQueue = tamanho da Input_Queue_Exec
CountAux = 1
Enquanto CountAux <= CountAntQueue faça
    LVT = Input_Queue_Exec.Timestamp
    TimestampAntQueue = Input_Queue_Exec.Timestamp
```

```
    LVTLinha = Input_Queue_Exec.LVTLinha
    CountAux = CountAux + 1  //Devolve para a Input_Queue_Exec
    Input_Queue_Exec.Timestamp = TimestampAntQueue
    Input_Queue_Exec.LVTLinha = LVTLinha
Fim Enquanto
// Retonar para Ready, com LVT e LVTLinha corretos
Ready.LVT = LVT
Ready.LVTLinha = LVTLinha
Retornar para Ready
```

Apêndice C

Publicações

Neste apêndice estão relacionadas as publicações realizadas, elaboradas com base nos trabalhos descritos nesta tese.

- SPOLON, R; SANTANA, M. J.; SANTANA, R. 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, 1999.
- SPOLON, R.; SANTANA, M. J.; SANTANA, R. H. C. A Methodology for Performance Evaluation of Optimistic Distributed Simulation Synchronisation Mechanisms. *Proceedings of the 14th European Simulation Multiconference (ESM'2000)*, p. 121-125, 2000.
- SPOLON, R.; ULSON, R. S.; SANTANA, M. J.; SANTANA, R. H. C. Simulação Distribuída, *Time Warp* e Variantes: Descrição e Análise Comparativa. Relatórios Técnicos do ICMC-USP, n° 111, Instituto de Ciências Matemáticas de São Carlos da Universidade de São Paulo. ISSN 0103-2569, 2000.
- FRANCÊS, C. R. L.; SPOLON, R.; SANTANA, M. J.; SANTANA, R. H. C. Modelagem e Especificação Utilizando Redes de Petri e Statecharts. Nota Didática 45 do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2000.

Apêndice D

Análises Estatísticas

O objetivo da inferência estatística é retirar conclusões sobre parâmetros populacionais a partir de análise dos dados amostrais. Alguns pontos importante são o tamanho amostral, o modo de seleção da amostra, a natureza da inferência desejada e a precisão das conclusões. Existem dois métodos para efetuar inferências, que envolvem o cálculo de intervalos de confiança e o uso do teste de hipóteses. A seguir ilustra-se a aplicação de intervalos de confiança neste trabalho (todos os resultados de saída foram obtidos a partir de 30 execuções diferentes da simulação *ALPHA/Sim*).

De acordo com o Teorema do Limite Central, para n (tamanho da amostra) suficientemente grande ($n \geq 30$), a distribuição da média amostral $\bar{y}$ é aproximadamente normal com $E(\bar{y}) = \mu$ e $V(\bar{y}) = \sigma^2/n$; $\bar{y}$ é considerado o melhor estimador para μ (Mendenhall; Sincich 1992).

O intervalo de confiança $(1 - \alpha)100\%$ (n suficientemente grande) para μ pode ser obtido com a fórmula da figura D.1. A interpretação do coeficiente de confiança $(1 - \alpha)$ é que se for construído um intervalo de confiança para cada amostra de tamanho n, obtidas a partir de uma população, espera-se que $(1 - \alpha)100\%$ dos intervalos englobem o verdadeiro valor do parâmetro em estudo.

$$\bar{y} \pm z_{\alpha/2}\sigma_{\bar{y}} = \bar{y} \pm z_{\alpha/2}\left(\frac{\sigma}{\sqrt{n}}\right) \approx \bar{y} \pm z_{\alpha/2}\left(s/\sqrt{n}\right)$$

Figura D.1: Intervalo de Confiança para a Média da População (Amostras Grandes).

A tabela D.1 exemplica o uso do cálculo de intervalo de confiança para o parâmetro de saída Comprimento Médio do *Rollback*, para a aplicação Servidor de Arquivos (α = 0,05). Todos os parâmetros de entrada foram mantidos padrões (como descrito no capítulo 6).

Tabela D.1: Intervalo de Confiança para o Comprimento Médio do *Rollback*.

Tempo de Execução (ms)	Comprimento Médio do *Rollback*	Intervalo de Confiança
30000	3,941247	3,941247 ± 0,001998
40000	3,943768	3,943768 ± 0,001898
50000	3,936051	3,936051 ± 0,002039