Renan Curvello Faria

Método para Análise de Troca de Protocolos Otimistas em um Ambiente de Simulação Distribuída.

Itajubá – MG

Dezembro de 2015

Renan Curvello Faria

Método para Análise de Troca de Protocolos Otimistas em um Ambiente de Simulação Distribuída.

Orientador:
Prof. Dr. Edmilson Marmo Moreira

Co-orientador:
Prof. Dr. Otávio Augusto Salgado Carpinteiro

Universidade Federal de Itajubá - UNIFEI
Instituto de Engenharia de Sistemas e Tecnologias da Informação
Engenharia da Computação

Itajubá – MG

Dezembro de 2015

"Se eu vi mais longe, foi por estar de pé sobre ombros de gigantes.
(Isaac Newton, Carta para Robert Hooke - 15 de Fevereiro de 1676)

Agradecimentos

Primeiramente a Deus, Nosso Senhor Jesus Cristo, pela vida que me concedeu, por iluminar os meus caminhos e pelas bênçãos concedidas durante a confecção deste trabalho por interseção da Santíssima Virgem Maria, de São Miguel Arcanjo e de todos os santos.

Ao professor Dr. Edmilson Marmo Moreira pela orientação, disponibilidade e solicitude na realização deste trabalho.

Ao professor Dr. Otávio Augusto Salgado Carpinteiro pela coorientação e pelas dicas úteis na realização deste trabalho.

Ao colega de mestrado Márcio Emílio Cruz Vono de Azevedo pela contribuição com a implementação do protocolo *Time Warp* e *Rollback* Solidário.

Ao colega de mestrado e doutorando Mateus Augusto Faustino Chaib Junqueira pelo auxílio com a plataforma LaTex.

Ao colega de mestrado Luiz Fernando Nunes pelo auxílio com a execução da aplicação no *cluster* do IESTI.

Aos meus pais Marcio e Rosana pelo amor, incentivo, compreensão e apoio constante.

À minha namorada Mariana pelo amor, carinho, companheirismo, compreensão e amizade.

À minha irmã Mariana pelo companheirismo, amizade e carinho.

Aos professores da UNIFEI, do IESTI e do ICE, por todo o conhecimento adquirido durante o transcorrer do curso de bacharelado em Ciência da Computação

e do curso de Mestrado em Ciência e Tecnologia da Computação, sem os quais a realização deste trabalho se tornaria impossível.

Ao GPESC e a UNIFEI pela oportunidade e pelos recursos oferecidos para a realização deste trabalho.

A CAPES pelo apoio financeiro.

Resumo

Este trabalho tem por finalidade, apresentar um método que seja capaz de determinar quando é vantajoso efetuar a troca entre dois protocolos de sincronização otimistas disponíveis na literatura: *Time Warp* e *Rollback* Solidário. Através da análise e comparação dos resultados entre intervalos de tempo previamente definidos do total de tempo da execução da simulação, é possível apresentar um estudo detalhado das características de ambos os protocolos. Os intervalos são definidos dados um limite máximo n para a execução total e o tamanho máximo para cada intervalo t, que definirão assim a quantidade de partições a serem analisadas (n/t). O método proposto neste trabalho utiliza-se de um algoritmo guloso, que irá analisar e comparar os resultados dos intervalos referentes a um mesmo espaço de tempo para os dois protocolos a fim de estabelecer um caminho de máxima eficiência da simulação, determinando uma possibilidade de execução mista, que fornece uma base de como o modelo se comporta mediante a troca de protocolos. Os parâmetros tomados como base de comparação entre os intervalos baseiam-se em métricas conhecidas na literatura e são usados na tomada de decisão sobre a troca dinâmica de protocolos. Embora as métricas utilizadas sejam significativas, no que diz respeito a se estabelecer um comparativo teórico entre os dois protocolos, a troca em si demandaria um alto custo para ser efetuada. Este custo e a possibilidade de compará-lo com o valor das métricas foram modelados neste trabalho.

Abstract

This work aims to present a method which is capable to determine define how advantageful is to execute the switch between two optimistic synchronization protocols available in literature: Time Warp and Solidary Rollback. Trough the analysis and comparison of results between time intersections previously defined on the total amount of time of the execution of the simulation it is possible to present a study in deepth and with thorough detailing of the specific characteristics of both protocols. The intervals are defined on an n maximum limit for the complete execution and the maximum size for each interval t, which thus define the number of partitions to be analyzed (n/t). The method proposed in this work makes use of a greedy algorithm, which will analyze and compare the results of intervals for the same period of time for the two protocols to establish a path of maximum simulation efficiency, determining a possibility of execution mixed, which provides a base upon how the model behaves exchange protocols. The parameters taken as a basis of comparison between the intervals are based on metrics known in the literature and are used in decision making on the dynamic exchange protocols. Although the metrics used are significant in relation to establish a theoretical comparison between the two protocols, the exchange itself would require a high cost to be made. This cost and the possibility of comparing it with the value of the metrics were modeled in this work.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API – *Application Programming Interface*

BPR – *Business Process Reengineering*

BTW – *Breathing Time Warp*

CGC – *Checkpoint* Global Consistente

CL – Chandy-Lamport

CMB – Chandy, Misra e Byrant

CT – Custo de Troca

FDAS – *Fixed Dependency After Send*

FIFO – *First In, First Out*

FR – Frequência de *Rollbacks*

GVT – *Global Virtual Time*

IaaS – *Infrastructure as a Service*

IDE – *Integrated Development Environment*

LM – Linha de Execução Média

LMI – Linha de Execução de Maior Incidência

LTW – *Local Time Warp*

LVT – *Local Virtual Time*

MPI – *Message Passing Interface*

MRIP – *Multiple Replication in Parallel*

NRAS – *No Receive After Send*

PaaS – *Plataform as a Service*

PDES – *Parallel Discrete-Event Simulation*

RDT – *Rollback-Dependency Trackability*

RE – *Rollbacks* por Eventos

SaaS – *Software as a Service*

SD – Sistema Distribuído

SNS – *Sync-and-Stop*

SO – Sistema Operacional

SRIP – *Single Replication in Parallel*

SSS – *Sparse State Saving*

TD – Tempo Desperdiçado

TMR – Tamanho Médio do *Rollback*

TMS – Tempo Médio da Simulação

TSIM – Tempo da Simulação

TT – Tempo de Transição

TTW – *Tentative Time Warp*

ZPF – *Z-Path Free*

1 Introdução

A simulação é uma ferramenta muito útil que permite que problemas do mundo real sejam solucionados através de modelos simplificados sem a necessidade da implantação efetiva do sistema real simulado.

A simulação computacional começou a ser utilizada efetivamente no começo da década de 60. A primeira área a utilizar este método como ferramenta de análise e planejamento foi a área militar nos EUA. Com o sucesso obtido na aplicação do método na área militar, sua extensão à indústria norte-americana ocorreu rapidamente, proporcionando o desenvolvimento do método através da evolução das linguagens de programação (CASTILHO, 2004).

Durante muitas décadas, até os dias atuais, simulações foram utilizadas para identificar comportamentos de sistemas complexos que envolvem diversas áreas científicas como: a Computação, a Física, a Matemática e a Química. Vários cientistas, através dos anos, utilizaram os resultados obtidos em simulações para tomar decisões referentes a projetos e pesquisas (BANKS et al., 2003). Atualmente, a simulação também é utilizada na área financeira para prever orçamentos e realizar investimentos.

O campo de aplicação da simulação é muito amplo estendendo sua aplicação para muito além das áreas puramente militar e acadêmica. Chwif e Medina (2014) dividem as áreas de aplicação em outros dois grandes setores, manufatura e serviços, e elencam alguns exemplos de aplicações da simulação nesses dois campos. Na área de serviços, simulações podem ser aplicadas em aeroportos, bancos, cadeias logísticas, *call centers*, escritórios, hospitais, parque de diversões, restaurantes, cadeias de *fast food* e supermercados. Na área de manufatura, a simulação pode

ser aplicada em vários pontos, tais como: sistemas de movimentação e armazenagem de materiais, linhas de montagem, células automatizadas, problemas de programação da produção e análise de estoques.

Simulações permitem a previsão de situações adversas com um alto nível de percepção, devido à quantidade de informações obtidas durante e ao final do experimento. Atualmente, as simulações são realizadas em sistemas computacionais visando agilidade na obtenção dos dados. Praticamente, todos os sistemas podem ser modelados, parcialmente ou totalmente, por equações matemáticas, cujo uso torna estes modelos úteis em simulações e permitem a obtenção de dados reais com baixo custo em relação à utilização de protótipos e com uma margem de segurança aceitável.

Muitas vezes, a simulação computacional sequencial pode requerer um tempo de execução inviável e para melhorar o tempo de resposta é possível distribuir a simulação entre diversos processadores de uma máquina paralela ou um sistema distribuído (SD). Tanenbaum e Stenn (2007) definem SD pelo conceito de transparência, no qual um conjunto de computadores independentes se apresenta a seus usuários como um sistema único e coerente. Em termos conceituais e abrangência, essa definição seria a proposta mais atual adicionada a características de tolerância a falhas.

A paralelização da simulação, no entanto, cria dificuldades que não existem na simulação sequencial como, por exemplo: a sincronização entre os processos executados em diferentes processadores, o balanceamento de carga e a sobrecarga na rede de comunicação. Estes dois últimos problemas podem ser contornados mediante o uso de técnicas de migração de processos, como mostrado em Junqueira (2012). O problema da sincronização de processos é mais complexo, pois a inexistência de um relógio físico global que sincronize as atividades de todos os processos envolvidos na simulação cria a necessidade de que estes sejam sincronizados através de relógios lógicos (LAMPORT, 1978). A sincronização de relógios físicos é inviável, pois, mesmo que se utilizem relógios de alta precisão e exatidão é impossível sincronizá-los perfeitamente, além disso, o custo em adquirir estes componentes é demasiadamente alto (TANENBAUM, 2010).

A simulação deve ser consistente, ter um início, meio e um fim, e gerar resultados confiáveis. Desta forma, a sincronização de processos é necessária para que, em uma simulação, não ocorram erros de causalidade: inconsistências relacionadas ao processamento de eventos na ordem cronológica errada. Em sistemas centralizados, as operações são executadas sequencialmente, logo, este tipo de erro não ocorre. Em sistemas distribuídos, e máquinas paralelas, a relação de causa e efeito pode ser violada gerando inconsistências nos resultados da simulação, devido ao paralelismo da execução que permite que as operações da simulação sejam executadas concorrentemente. A sincronização de processos deve ser baseada na ordem cronológica do processamento dos eventos, e não em um relógio físico sincronizado, para que os resultados da simulação possam ser considerados confiáveis (CHANDY; MISRA, 1979).

Com a evolução dos estudos envolvendo sistemas distribuídos, duas classes de protocolos de sincronização de processos surgiram com o propósito de garantir a consistência dos resultados das simulações: os protocolos conservativos e os protocolos otimistas. Os protocolos conservativos não permitem a ocorrência de erros de causa e efeito prevenindo-os de acontecerem, ao passo que os protocolos otimistas permitem a ocorrência dos erros de causa e efeito, vindo a tratá-los logo após a sua identificação (WANG; TROPPER, 2007).

Os primeiros protocolos de sincronização se basearam em uma abordagem conservativa (FUJIMOTO, 2003) e foram propostos por Chandy e Misra (1979) e Bryant (1977). A junção destes algoritmos ficou conhecida com a sigla CMB, que corresponde às iniciais dos nomes dos autores. Estes protocolos caracterizam-se por evitar que erros de causa e efeito ocorram durante a execução da simulação. Desta forma, um evento só pode ser tratado se for garantido pelo protocolo de sincronização que nenhum outro evento com um *timestamp* menor que o dele será executado no futuro.

Em contrapartida, os protocolos de sincronização otimistas não impedem estes erros de ocorrerem, permitindo assim que os processos executem eventos livremente e sem restrições, o que traz, evidentemente, um ganho em termos de desempenho na execução da simulação. Assim, caso um erro de causalidade aconteça, o protocolo utiliza um mecanismo que recupera o sistema e mantém consistência do

mesmo, recuperando a computação até um estágio anterior à falha detectada no sistema. Este mecanismo chama-se *rollback* e assemelha-se à operação de *rollback* que ocorre em banco de dados quando uma falha de consistência ameaça a integridade do banco. Segundo Wang e Tropper (2007), o *rollback* dos protocolos de sincronização otimistas possui a função de retomar a computação de um ponto específico, reprocessar os eventos corretamente e manter a consistência do sistema. Este trabalho foca em duas abordagens para protocolos otimistas conhecidos na literatura que lidam com este problema de maneira distinta. Uma deles é o *Virtual Time*, cuja implementação para simulação distribuída se chama *Time Warp* (JEFFERSON, 1985), e a outra faz o uso de *checkpoints* globais consistentes, abordagem implementada no *Rollback* Solidário (MOREIRA, 2005).

1.1 Motivação

A simulação tem sido largamente empregada devido à sua flexibilidade e baixo custo possibilitando analisar o desempenho de um sistema real, através da execução de modelos que o representem, antes que o investimento na compra de *hardware* e *software* seja realizado. Simulações são realizadas principalmente em sistemas computacionais, que podem possuir configuração centralizada ou distribuída, visando agilidade na obtenção dos dados.

Tanto o protocolo conservativo CMB, quanto os protocolos otimistas *Time Warp* e *Rollback* Solidário e suas variantes, possibilitam gerenciar a ocorrência de erros de causa e efeito em uma simulação distribuída de modo a conservar sua consistência. Diversos estudos envolvendo o aprimoramento destes protocolos, até então, foram desenvolvidos. Lobato (2001) apresenta diversas otimizações tanto do CMB, a exemplo do *Local Adaptive Protocol*, quanto do *Time Warp*, a exemplo do *Breathing Time Warp* (BTW), *Local Time Warp* (LTW) e *Tentative Time Warp* (TTW). Junqueira (2012) apresentou dois mecanismos para a migração de processos dentro de uma simulação distribuída, ao passo que Carvalho (2013) avaliou diferentes métricas de escalonamento para o protocolo *Time Warp*. Barbosa (2012) propôs um ambiente gráfico para a modelagem e monitoração de processos em uma simulação distribuída, tomando como base o *framework* desenvolvido por

Cruz (2009). Além de mudanças no modo de funcionamento do mecanismo de *rollback*, foram feitos estudos a respeito de aspectos laterais do *Time Warp* quanto ao salvamento de estados (LAPRE et al., 2015), cálculo do tempo global da simulação (LIN; YAO, 2015) e a proposição de esquemas de rejuvenscimento de software para uma simulação executada pelo *Time Warp* (FUKUMOTO; OHARA, 2015).

Morselli (2000) e Kawabata (2005) propuseram diferentes mecanismos de troca dinâmica entre os protocolos conservativo, CMB, e otimista, *Time Warp*. Entretanto, nenhum mecanismo de troca de protocolos otimistas, entre o *Rollback* Solidário e *Time Warp*, havia sido, até então, proposto na literatura, e nenhuma tentativa de compará-los a fundo, considerando suas peculiaridades, havia sido feita. O método implementado neste trabalho toma como base a implementação dos protocolos em questão, e do gerador de modelos, disponibilizados em Azevedo (2012), além de aprimorar o comparativo entre eles.

1.2 Objetivos

O objetivo desta dissertação de mestrado é propor um método que determine quando é vantajoso efetuar a troca entre dois protocolos otimistas, *Time Warp* e *Rollback* Solidário, através da análise e comparação dos resultados de desempenho entre intervalos de tempo previamente definidos do total de tempo da execução da simulação. A partir desta análise é possível apresentar um estudo detalhado das características dos dois protocolos em questão. Para este fim, foi implementado um mecanismo de troca de protocolos otimistas.

Morselli (2000) define três etapas para a troca de protocolos de sincronização:

1. Detecção da necessidade da troca de protocolos;
2. Garantia da manutenção da consistência da simulação no momento da troca;
3. Adaptação das estruturas de dados para possibiltar a troca de protocolos de sincronização.

O método proposto neste trabalho, diferente dos trabalhos anteriormente ci-

tados, substitui a detecção dinâmica da necessidade da troca de protocolos, por possibilidades fixas, delimitadas a partir da transição dos intervalos de tempo, definidos no início da execução da simulação. A manutenção da consistência da simulação durante a troca, é garantida pelo mecanismo de cálculo do *Global Virtual Time* (GVT) ao fim de cada intervalo, ao passo que as técnicas de programação orientada a objetos facilitam a adaptação das estruturas de dados sem grande esforço.

É importante salientar, que a característica semi-síncrona do procedimento de *rollback* do protocolo *Rollback* Solidário torna a troca parcial de protocolos, otimização definida no trabalho de Kawabata (2005), complicada de ser tratada em termos deste protocolo. Desta forma, a coexistência do *Rollback* Solidário com qualquer outro protocolo de sincronização, durante um mesmo instante de tempo em uma simulação, se torna muito complexa.

As métricas usadas na comparação entre intervalos de tempo equivalentes foram adaptadas do trabalho de Lobato (2001) e usadas na tomada de decisão sobre a troca de protocolos. Embora estas sejam altamente significativas, no que diz respeito a se estabelecer um comparativo teórico entre os dois protocolos otimistas, existe um custo associado à troca de protocolos que precisa ser levado em consideração. Este custo foi mensurado neste trabalho, de modo que este possa ser comparado com o resultado obtido por meio das métricas.

Embora este trabalho disponha de um mecanismo de troca de protocolos, implementado como sugestão de trabalhos futuros de Azevedo (2012), o objetivo deste trabalho não é, a priori, construir um mecanismo de troca dinâmica de protocolos durante a simulação.

Em suma, o objetivo principal deste trabalho é definir um método que: (1) possa dizer com certo grau de confiabilidade, em quais condições esta troca seria vantajosa, (2) conferir se um modelo teórico de troca de protocolos estabelecido por meio de métricas de desempenho converge na prática por meio de estudo de casos, e, em última análise, (3) verificar se existe alguma correlação, em termos de melhorias de desempenho, entre a escolha de um dos dois protocolos com o número de processos de um modelo de simulação, e também do número de canais

de comunicação inter-processo que o compõe.

1.3 Estrutura da Dissertação

Com este resumo geral sobre simulação, a motivação e os objetivos deste trabalho, serão apresentados, nos próximos capítulos, além dos conceitos preliminares sobre Simulação de Eventos Discretos, Simulação Distribuída, Protocolos de Sincronização Conservativos e Otimistas, uma revisão de literatura detalhada a respeito dos protocolos *Time Warp* e *Rollback* Solidário (capítulo 2). O método para identificação de um caminho mínimo entre intervalos de tempos pré-definidos de uma simulação distribuída (capítulo 3), alguns detalhes sobre a implementação deste método (capítulo 4), e alguns estudos de casos para a aplicabilidade deste método (capítulo 5). Finalmente, o capítulo seis apresenta as conclusões finais deste trabalho.

2 Simulação Distribuída e Computação Paralela

Neste capítulo é feita uma revisão bibliográfica sobre alguns conceitos sobre simulação, protocolos otimistas, *Time Warp* (JEFFERSON, 1985), *Rollback* Solidário (MOREIRA, 2005) e a biblioteca de comunicação MPI (*Message Passing Interface*). Os protocolos e essa biblioteca de passagem de mensagens são essenciais na implementação do mecanismo *Protocol Swapper*, descrito no capítulo 3 e implementado no capítulo 4. A implementação dos protocolos na linguagem Java, usando a biblioteca MPJ-Express, são herdadas do trabalho de Azevedo (2012), e a eles foram acrescidos mecanismos para o cálculo de GVT (*Global Virtual Time*), sendo estes mais bem detalhados nos capítulos 3 e 4.

Ambos os protocolos são otimistas e não impedem a ocorrência de eventos de causa e efeito que podem ser gerados após o recebimento de mensagens *stragglers*. Por definição, mensagens *stragglers* são mensagens que deveriam ser processadas antes do estado atual do processo que a recebeu. Desta forma, quando uma destas mensagens chega a um determinado processo, esta deve ser tratada através de procedimentos de *rollback* (ZIMMERMAN; KNOKE; HOMMEL, 2006).

A principal diferença entre os dois protocolos está na forma como cada um trata uma mensagem *straggler* quando esta é recebida por um processo da simulação. O procedimento de *rollback* do *Time Warp* é totalmente diferente do procedimento utilizado no *Rollback* Solidário. Enquanto o primeiro usa a abordagem de *rollback* em cascata para desfazer a computação errônea, o segundo usa um mecanismo baseado na noção de corte global consistente (BABAOGLU; MARZULLO, 1993) para efetuar o retorno da computação para um estado consistente. O capítulo 3 esta-

belece um arcabouço teórico para embasar uma possível comparação entre os dois protocolos.

2.1 Conceitos Básicos

As seções seguintes estendem-se sobre alguns conceitos básicos, e apresentam uma revisão bibliográfica, antes de se falar dos protocolos *Time Warp* e *Rollback* Solidário mais especificamente.

2.1.1 Simulação

Existe uma gama de definições a que se pode atribuir à palavra simulação, e vários autores na literatura atribuem um significado distinto, mas que basicamente gravitam em torno de um eixo comum, o qual pode ser expresso na definição de simulação como sendo a execução e análise de resultados de modelos. Estes são abstrações simplificadas de um sistema, a fim de avaliar a priori o desempenho de um sistema segundo parâmetros que o caracterizem para os fins desejados, e podem ser representados como Statecharts (HAREL, 1987), Redes de Petri (MURATA, 1989), Rede de Filas (COSTA, 2006), entre outros.

Segundo Banks (1999), uma simulação pode ser definida como uma imitação de um processo do mundo real sobre o tempo, caracterizada pela geração de um histórico artificial do comportamento de algum sistema. A observação deste histórico possibilita ao pesquisador ou analista criar inferências concernentes às características operacionais, analisar o comportamento e formular questões sobre o sistema real simulado. A geração destes históricos é realizada através de modelos de sistemas criados especificamente para descrever um sistema particular.

Um modelo é uma abstração da realidade aproximando-se do verdadeiro comportamento do sistema, mas sempre mais simples do que o sistema real. Modelos devem ser complexos o suficiente para responder as questões levantadas, mas não demasiadamente complexos (BANKS, 1999). Desta forma, o modelo construído deve apresentar uma complexidade menor do que o sistema real, pois, caso contrá-

rio, obtêm-se um problema e não um modelo propriamente dito (CHWIF; MEDINA, 2014).

Segundo Perin Filho (1995), o modelo de um sistema é uma representação simplificada de um conjunto estruturado de componentes que interagem de modo regular entre si e com o meio ambiente, sendo classificado de várias maneiras de acordo com suas características.

Neste contexto, a simulação surge como uma alternativa viável na forma de se estudar um sistema real por intermédio de modelos matemáticos que o represente, conforme ilustra a figura 2.1.

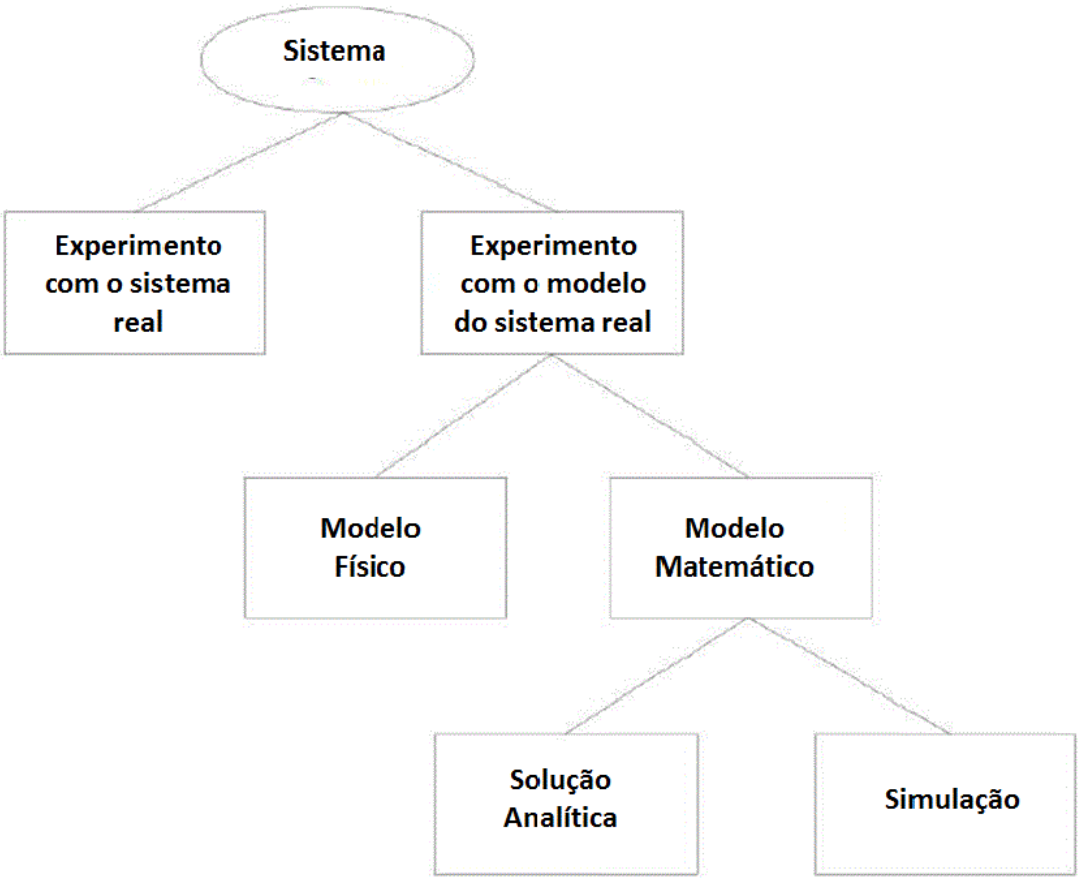

Figura 2.1: Modos de se estudar um sistema (BABULAK; WANG, 2010)

Um modelo que represente um sistema real definirá o tipo de simulação a ser adotada e executada computacionalmente para a obtenção de resultados que des-

crevam o comportamento desse mesmo sistema. A análise estatística dos resultados gerados pode contribuir na tomada de decisões.

2.1.1.1 Tipos de Simulações e Modelos

Inicialmente, os modelos de simulação podem ser classificados como determinísticos ou estocásticos. Os modelos determinísticos são aqueles que não contêm nenhuma variável, ou seja, para um conjunto conhecido de dados de entrada tem-se um único conjunto de resultados de saída. Os modelos estocásticos possuem uma ou mais variáveis aleatórias como entrada, que levam a saídas aleatórias, e que são utilizadas quando pelo menos uma das características operacionais é dada por uma função de probabilidade. Os modelos ainda podem ser classificados em estáticos ou dinâmicos. Os modelos estáticos visam representar o estado de um sistema em um instante ou que, em suas formulações, não se leva em conta a variável tempo. Em contrapartida, os modelos dinâmicos são formulados para representarem as alterações de estado do sistema ao longo da contagem do tempo de simulação (BABULAK; WANG, 2010).

Nance e Sargent (2002) categorizam os modelos de simulação em discretos e contínuos, sendo os modelos discretos aqueles em que os estados mudam discretamente no tempo, ao passo que os modelos contínuos são aqueles aos quais seus estados mudam continuamente no tempo. Os modelos discretos determinam o uso da simulação de eventos discretos para sua execução, ao passo que os modelos contínuos determinam o uso da simulação contínua para a sua execução.

A figura 2.2 ilustra uma taxonomia relacionando todas as classificações de tipos de modelos de simulação apresentados: estocástico, determinístico, estático, dinâmico, contínuo e discreto.

Chwif e Medina (2014) classificam as simulações em computacionais e não computacionais, sendo a simulação computacional em três categorias básicas: simulação de “Monte Carlo”, contínua e de eventos discretos. A simulação de Monte Carlo utiliza-se de geradores de números aleatórios para simular sistemas físicos e matemáticos, nos quais não se considera o tempo explicitamente como uma variável. A simulação contínua e a simulação de eventos discretos levam em consideração

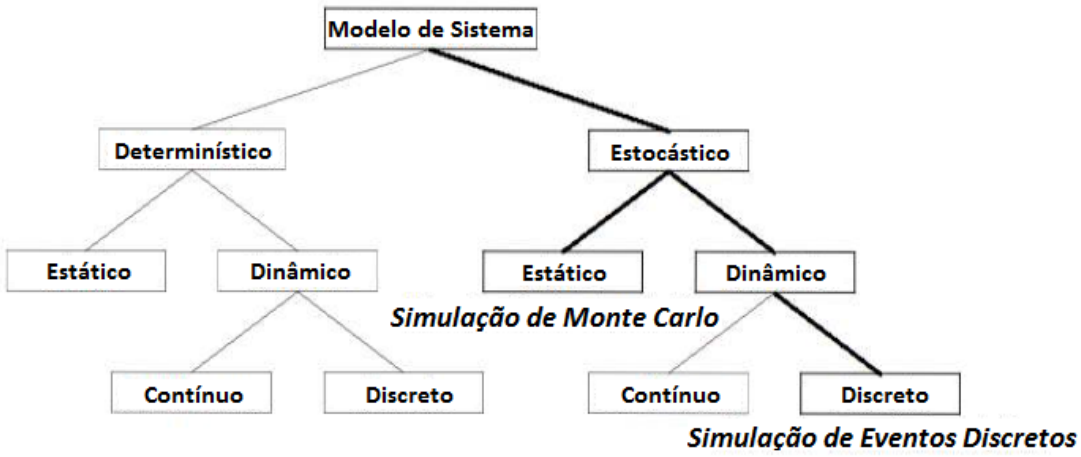

Figura 2.2: Taxonomia de modelos de simulação (BABULAK; WANG, 2010)

a mudança de estado do sistema ao longo do tempo, sendo a primeira utilizada para modelar sistemas cujo estado varia continuamente no tempo, ao passo que a segunda é utilizada para modelar sistemas que mudam o seu estado em momentos discretos do tempo, a partir da ocorrência de eventos. Mello (2001) ainda discorre sobre modelos combinados, nos quais as variáveis dependentes podem variar discretamente, continuamente, ou continuamente com saltos discretos superpostos, podendo a variável tempo pode ser discreta ou contínua.

A simulação de eventos discretos pode ser definida como aquela que contém atividades que avançam discretamente no tempo, ou seja, é a simulação usada para estudar sistemas onde as mudanças de estado são discretas (JAGTAP; ABU-GHAZALEH; PONOMAREV, 2012). O uso da simulação computacional, para sincronização e comunicação inter-processos, refere-se exclusivamente à simulação de eventos discretos. Os modelos discretos são mais adequados para representar sistemas computacionais, devido ao comportamento discreto destes, tornando a simulação desse tipo de sistema mais flexível. Não obstante, a partir deste ponto, e para maior facilidade de entendimento da proposta deste trabalho, toda e qualquer referência à palavra “simulação” deverá ser subentendida como a simulação computacional de eventos discretos.

2.1.1.2 Simulação Paralela e Simulação Distribuída

Fujimoto (2000) cunha o termo "simulação paralela de eventos discretos" (PDES - *Parallel Discret-Event Simulation*) em referência a execução dos programas responsáveis pela simulação em plataformas computacionais de múltiplos processadores. A divisão da computação da simulação em outras sub-computações concorrentes reduz o tempo de execução por um fator quase correspondente ao número de processadores utilizados, só não sendo igual devido ao *overhead* oriundo da comunicação inter-processos. Cada parte do modelo simulado é implementada em programa visando sua redução de complexidade (CHWIF; PAUL; BARRETTO, 2006) e alocada a um processador do SD ou da máquina paralela.

Segundo Fujimoto (2003), historicamente, o termo "simulação paralela" refere-se à execução da simulação em um sistema de computador acoplado rigidamente, como, por exemplo, um supercomputador ou um multiprocessador de memória compartilhada, ao passo que, o termo "simulação distribuída" está vinculado às simulações distribuídas geograficamente. A diferença principal entre esses dois conceitos se dá por uma questão de foco e objetivos. A simulação paralela visa reduzir do tempo de execução de uma simulação, focando em sistemas computacionais de alto desempenho, ao passo que, a simulação distribuída aborda questões mais amplas como a reutilização de simulações já existentes e a exploração de recursos, como pessoas e equipamentos, que estão geograficamente distribuídos (YILMAZ et al., 2014).

Entretanto, com o surgimento de novos paradigmas computacionais, como *Cluster Computing* e *Grid Computing*, essa divisão tem se tornado cada vez mais tênue. O paradigma recente de *Cloud Computing* propõe a computação como um serviço, similar ao que seria o serviço de eletricidade, gás ou água, e torna essa divisão ainda mais tênue, pois, a preocupação com os serviços de infraestrutura (IaaS - *Interface as a Service*), plataforma (PaaS - *Plataform as a Service*) e *software* (SaaS - *Software as a Service*) são delegados aos provedores da *cloud*, o que traz ainda mais transparência à simulação e reduz, pelo menos aparentemente, a distinção entre uma simulação executada em uma máquina paralela e a simulação executada em um ambiente em nuvem (FUJIMOTO; MALIK; PARK, 2010). Além

disso, Taylor et al. (2012) e Yilmaz et al. (2014) apontam como tendências futuras, outras áreas e tecnologias, nas quais o uso da PDES pode se aprimorar tais como: computação ubíqua, supercomputação, megadados (*Big Data*), sistemas adaptativos complexos, redes sem escala, entre outros.

Uma implementação de simulação distribuída encontra obstáculos que não existem em sistemas centralizados, tais como, sincronização dos processos, balanceamento de carga e a sobrecarga na rede de comunicação. Em um ambiente distribuído, os processos lógicos devem ser executados de maneira que eventuais erros de causa e efeito não afetem a corretude do resultado da simulação, ou seja, a execução de eventos fora da ordem natural que seria obtida em uma simulação sequencial (YILMAZ et al., 2014). Para que isso ocorra, cada processo lógico tem seu próprio relógio lógico, que indica o progresso da simulação e, como não há um ambiente de memória compartilhada entre eles, as informações entre os processos são trocadas através de mensagens.

Uma maneira de garantir a ordem de execução cronológica dos eventos é utilizando protocolos que controlam o sincronismo entre os processos da simulação.

2.1.2 Protocolos de Sincronização Distribuída

Um processo lógico que contenha partes do modelo simulado possui suas variáveis de estado, as quais algumas armazenam uma parte dos resultados de simulação, e que por sua vez definem o estado local do processo. Denota-se por estado local de um processo, o estágio salvo do valor das variáveis de estado de um processo após a execução de um ou mais eventos durante a execução do processo lógico (BABAOGLU; MARZULLO, 1993). Uma computação distribuída pode ser representada por vários desses processos lógicos. O conjunto de todos os estados locais, dos processos que representam a computação distribuída, caracteriza o estado global do sistema (ZIMMERMAN; KNOKE; HOMMEL, 2006).

A simulação progride de acordo com a execução de eventos pelos processos lógicos que compõem o modelo. Deste modo, cada processo possui uma fila de eventos futuros, internos e externos, que devem ser processados durante a execu-

ção da simulação, e que são armazenados em ordem cronológica de acordo com uma marca de tempo inerente a cada processo, chamada *timestamp*. Conforme a simulação vai avançando no tempo, o evento com o menor *timestamp* é retirado do início da fila e processado.

À medida que os eventos vão sendo retirados da fila de eventos futuros, o relógio lógico do processo que contém o evento retirado avança até o *timestamp* deste evento, dando, desta forma, andamento à simulação. A ordem cronológica do evento é preservada, pois o evento retirado do início da fila é o que possui o menor *timestamp*.

Uma simulação distribuída, para ter validade, requer a não ocorrência de erros de causa e efeito durante a simulação. Para tanto, a relação de precedência causal de Lamport (1978) deve ser respeitada pelos eventos. Esta relação de ordem parcial pode ser resumida do modo como se segue:

Dado dois eventos, e_1 e e_2, e_1 ocorre antes de e_2 se:

- O *timestamp* de e_1 for menor que o *timestamp* de e_2;
- e_1 seja processado antes de e_2.

Duas abordagens de protocolos de sincronização para lidar com os erros de causa e efeito e, consequentemente, assegurar a relação de precedência causal de Lamport (1978), são os protocolos conservativos e os protocolos otimistas, evitando ou corrigindo erros de causa e efeito, respectivamente (PIENTA; FUJIMOTO, 2013).

2.1.2.1 Protocolos Conservativos

Os protocolos conservativos evitam que erros de causalidade ocorram durante a execução da simulação, impondo aos processos lógicos uma sincronização explícita (PARK; FUJIMOTO; PERUMALLA, 2004). Algumas características são necessárias a esses protocolos para garantir que erros de casualidade sejam evitados:

- Um processo só pode se comunicar com outros processos se existirem, entre cada par de processos, um canal confiável previamente definido antes da

simulação iniciar. Desta forma, os canais de comunicação entre os processos são estáticos;

- A ordem cronológica da execução é mantida, pois as mensagens que trafegam pelos canais de comunicação são armazenadas em uma fila de ordem FIFO (*First in First Out*);

- Todas as mensagens que trafegam entre os canais de comunicação devem estar rotuladas com o relógio local do processo (LVT) que a enviou.

Uma grande desvantagem no uso deste tipo de protocolo de sincronização é a possibilidade da ocorrência de *deadlock* entre os processos. Há duas abordagens para o tratamento de *deadlocks* em sistemas conservativos, as quais uma consiste em prevenção, e a outra na detecção e reparação de *deadlocks* (CHANDY; MISRA, 1981). Entretanto, esta última é mais adequada para o tratamento de *deadlock*, pois a abordagem de prevenção acarreta a desvantagem da sobrecarga nos canais de comunicação (BUI; LANG; WORKMAN, 2003), embora este problema possa ser minimizado (MOREIRA, 2005).

Misra (1986) propõe o conceito de *lookahead*, que consiste em determinar um intervalo de tempo no qual todos os processos não possam agendar eventos futuros, a fim de assegurar que neste intervalo existe a certeza absoluta que a execução destes eventos não causará erros de causa e efeito. Esta otimização proporcionou uma redução considerável no tempo de execução deste protocolo.

Para um estudo mais aprofundado deste tipo de protocolo, outras variações das técnicas apresentadas acima podem ser encontradas em: SRADS (REYNOLDS, 1982), Appointments (NICOL; REYNOLDS, 1984), Turner Carriernull Scheme (CAI; TURNER, 1990) e SPaDES/Java (TEO; NG, 2002). Otimizações visando melhorias deste protocolo em termos de desempenho podem ser encontradas em Meyer e Bagrodia R (1998), Deelman (2001), Chen e Szymanski (2002), Chen e Szymanski (2002b), entre outros.

2.1.2.2 Protocolos Otimistas

Ao contrário dos protocolos conservativos que não permitem a ocorrência de erros de causa e efeito, os protocolos otimistas permitem que esses erros ocorram.

Os registros necessários à sincronização entre os processos e o controle cronológico do sistema são o LVT, o *timestamp* e o GVT. O LVT é um contador de eventos que registra a ordem cronológica da computação de cada processo que modifica seu valor a cada ocorrência de um evento no processo, seja ele interno ou externo. O *timestamp* indica ao processo quando um evento deve ser executado. O GVT é representado pelo menor LVT do sistema e indica que nenhum processo irá criar eventos com o *timestamp* com valor inferior ao do GVT.

A identificação de erros de causa e efeito ocorre quando uma mensagem, cujo *timestamp* seja menor que o LVT do processo receptor, é recebida e subsequentemente é realizada a comparação entre os valores do LVT do processo receptor e o *timestamp* da mensagem. O protocolo então fará o *rollback* retrocedendo a computação, mediante a recuperação de estados salvos, para um evento cujo LVT seja de valor igual ou menor que o valor do *timestamp* da mensagem recebida. As mensagens que causam *rollback* em um sistema com protocolo otimista são chamadas de mensagens desgarradas ou mensagens *stragglers*. A Figura 2.3 ilustra a chegada de uma mensagem *straggler* no processo, com as setas na horizontal representando o processo em execução no decorrer do tempo, e os quadrados representando os eventos, processados e não processados. Nesta situação, os eventos com LVTs iguais a 12, 22 e 35 já estariam processados, ao passo que o evento a ser processado após a chegada da mensagem *straggler*, com *timestamp* 17, e o evento com LVT 35 ainda necessitariam ser processados. Desta forma, o sistema precisaria fazer *rollback* e retornar a computação para o evento 12, processar o evento causado pela mensagem *straggler* com *timestamp* 17, desfazer os eventos 22 e 35 colocando-os novamente na fila de eventos futuros e reprocessa-los no tempo lógico certo.

No caso ilustrado na figura 2.3 é possível visualizar como o procedimento do *rollback* aconteceria para um único processo, entretanto, em uma simulação distribuída, tudo que afeta um processo pode, e geralmente vai, afetar os outros demais

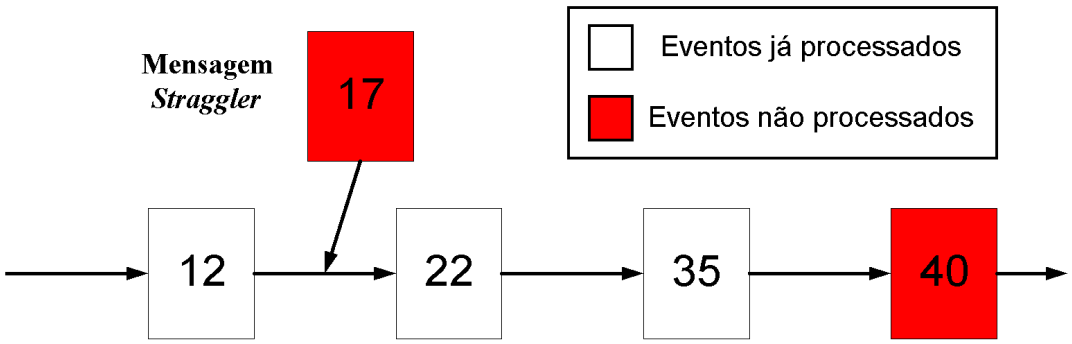

Figura 2.3: Mensagem *Straggler* (MOREIRA, 2005)

processos envolvidos na simulação. O *rollback* não pode se suceder apenas em um processo sem levar em consideração o modo como isso afetará os demais processos com os quais trocam mensagens e possuem uma relação de causalidade, sob o risco de comprometer os resultados da simulação e induzir erros de causa e efeito em outros processos. Quando um processo vai fazer *rollback*, ele precisa verificar se o sistema voltará para um estado global consistente.

2.2 Time Warp

Apresentado o *modus operandi* básico dos protocolos otimistas, o tratamento de um erro de causa e efeito por meio de uma operação de *rollback*, esta seção e a seção 2.3 apresentam, respectivamente, detalhes dos protocolos *Time Warp* e *Rollback* Solidário.

No protocolo *Time Warp*, quando uma mensagem *straggler* é recebida, o processo que a identificou realiza *rollback* imediatamente, para que o sistema se mantenha consistente. Este processo ao recuperar sua computação, deve identificar todas as mensagens que foram enviadas durante este período e enviar antimensagens para que os processos que receberam alguma mensagem durante este intervalo também corrijam seus estados. Para tanto, o processo receptor deve: (1) realizar o procedimento de *rollback*, retornando até o último estado salvo com LVT menor que o *timestamp* da mensagem *straggler* recebida; (2) verificar se alguma mensagem foi enviada para outros processos durante o intervalo de retorno e (3) disparar as

respectivas antimensagens, para que a computação retroceda em outros processos ligados ao processo que sofreu o *rollback* inicialmente.

Os *rollbacks* gerados por mensagens *stragglers* são denominados *rollbacks* primários, ao passo que os *rollbacks* gerados por antimensagens são denominados *rollbacks* secundários (FUJIMOTO, 2000). *Rollbacks* primários podem gerar *rollbacks* secundários em outros processos por meio de antimensagens (JEFFERSON, 1985; RAJAEI, 2007), ao passo que esses últimos podem gerar outros *rollbacks* secundários em outros processos de modo análogo. Esse procedimento é chamado de *rollback* em cascata e garante o retorno da simulação para um estado consistente (BABAOGLU; MARZULLO, 1993) após os efeitos de cada *rollback* primário serem totalizados na simulação, apesar de poderem fazer com que a execução dos eventos recue até o ponto inicial da computação, o que pode trazer uma grande degradação de desempenho da simulação (RAJAEI, 2007).

Para que o procedimento de recuperação do sistema consiga retornar a simulação para um estado consistente, no qual o sistema possa continuar executando sem alteração na corretude dos resultados gerados, cada processo utiliza duas filas de armazenagem de eventos:

- Fila de eventos futuros: nesta fila são armazenados os eventos recebidos pelas mensagens inerentes ao modelo simulado, as quais devem ser processadas em um futuro próximo, especificado pelos *timestamps* das mensagens recebidas.
- Fila de antimensagens: nesta fila são armazenadas todas as mensagens enviadas para serem utilizadas em caso de *rollback*.

Com o decorrer da simulação, o tamanho das duas filas de armazenamento de dados pode ultrapassar os limites de armazenamento de dados do sistema. Essa situação é conhecida como *overflow*, isto é, quando a estrutura de dados excede sua capacidade de armazenamento, fazendo com que o sistema entre em colapso. Para a solução deste problema, é necessário um mecanismo de "coleta de lixo", que descarte os eventos que não serão mais utilizados pelo sistema.

As subsessões seguintes tratam de detalhamentos a respeito de algumas características desse protocolo.

2.2.1 Antimensagens

Como antimensagens, pode-se entender as cópias das mensagens enviadas pelos processos para seus pares, de modo que, estas possam ser utilizadas pelo mecanismo de *rollback* do protocolo, para se desfazer a computação dos processos que receberam mensagens do processo que sofreu *rollback* devido à uma mensagem *staggler*. Desta forma, cada processo armazena uma lista de antimensagens: cada mensagem enviada gera sua respectiva antimensagem. Quando um processo sofre *rollback* primário, este envia suas antimensagens para os demais processos, os quais havia enviado mensagens anteriormente, no intervalo anterior ao valor autal do LVT e posterior ao valor do LVT após o procedimento de *rollback*.

As antimensagens possuem por finalidade a eliminação de mensagens das listas dos processos envolvidos em um *rollback*. A cada operação de *rollback*, o processo que o desencadeou verifica se alguma mensagem deve ser eliminada e, caso seja necessário, o processo reenvia a mensagem novamente. O processo receptor, ao receber esta antimensagem, passa a ter duas mensagens idênticas, porém com sinal oposto, na sua fila de entrada. Não obstante, essas mensagens devem ser eliminadas. Se o processo receptor já processou a mensagem em questão, e enviado outras quando da chegada da antimensagem, o mesmo deverá efetuar *rollback* e realizar o procedimento de envio de antimensagens descrito (ZENG; CAI; TURNER, 2004) .

Na proposta original de Jefferson (1985) é considerada a possibilidade da antimensagem ser recebida pelo processo receptor antes da sua mensagem correspondente, e sempre é tomado como pressuposto que todas as mensagens recebidas vão para fila de mensagens recebidas ou fila de eventos futuros a serem executados. Na implementação do *Time Warp* usada nesse trabalho, essas questões são contornadas, o que gera uma facilidade no tratamento das antimensagens.

A figura 2.4 ilustra, em linhas gerais, o funcionamento do protocolo *Time Warp*. Uma simulação distribuída, de quatro processos, é apresentada em um diagrama espaço-tempo, uma forma conveniente de se visualizar uma computação distribuída (SCHWARZ; MATTERN, 1994). As linha horizontais setadas representam a evolução dos tempos lógicos dos processos, ao passo que os círculos em preto

representam os eventos processados pelo sistema, com o índice subscrito de $e_{i,j}$, contendo o *id* do processo (i), separado por vírgula do *id* do evento (j). As setas contínuas representam mensagens inerentes da simulação de um processo para o outro, ao passo que as setas tracejadas representam antimensagens durante o procedimento de *rollback*, desencadeado por uma mensagem *staggler* com *timestamp* igual a 12, representada por uma linha sinuosa que sai do processo P_3 e se liga ao processo P_1. Após a realização do *rollback*, a computação retorna para um estado consistente indicado na figura pelo corte $C = \{e_{1,2}, e_{2,2}, e_{3,2}, e_{4,2}\}$

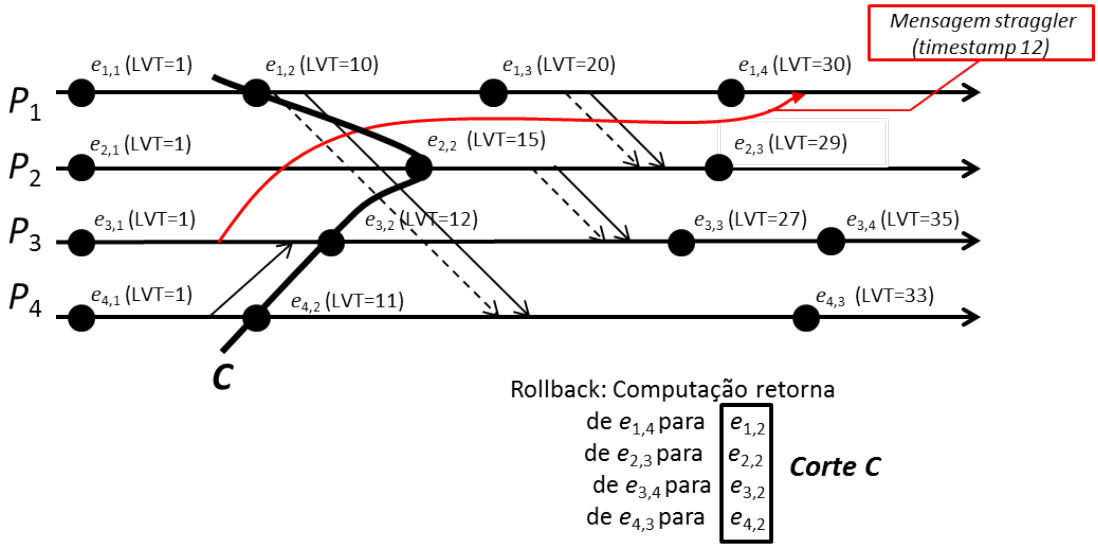

Figura 2.4: Protocolo *Time Warp* em linhas gerais

2.2.2 Cancelamento de Mensagens

As antimensagens são de fundamental importância no protocolo *Time Warp* pois, o mecanismo de *rollback*, responsável por desfazer computações quando da ocorrência de erros de causa e efeito, depende exclusivamente delas.

Um fator importante a ser considerado na implementação do protocolo é como o sistema deve se comportar na necessidade do envio de antimensagens. Para tanto, existem três estratégias utilizadas para o cancelamento dos eventos processados, ou não, durante a recuperação do sistema (GAFNI, 1988; REIHER et al., 1990; ISKRA; ALBADA; SLOOT, 2003):

- Cancelamento Agressivo: em que todas antimensagens são enviadas imediatamente para os demais processos que receberam mensagens do processo que realiza o *rollback*.
- Cancelamento Preguiçoso: em que as antimensagens só são enviadas se a nova execução da computação após o procedimento de *rollback* não coincide com a computação anterior.
- Cancelamento Dinâmico: em que o sistema decide qual das estratégias de cancelamento deve ser utilizada.

2.2.3 Salvamento de Estados

Para que o protocolo recupere uma computação após um procedimento de *rollback*, é necessário que exista um mecanismo de salvamento dos estados da simulação, que registre as variáveis da simulação que caracterizem seu andamento, e possa restaurar seus valores, caso algum erro de causalidade ocorra.

Dentre estes mecanismos, desenvolvidos para atender as necessidades de salvamento de estados dos protocolos otimistas, estão o *Copy State Saving* (JEFFERSON, 1985), *Incremental State Saving* (STEINMAN, 1993; BAUER; SPORRER, 1993; RONNGREN et al., 1996; WEST; PANESAR, 1996), *Sparse State Saving* (QUAGLIA, 1998) (também conhecido como *Infrequent State Saving* (LIN et al., 1993) ou ainda *Periodic State Saving* (PREISS; MACINTYRE; LOUCKS, 1992; FLEISCHMANN; WILSEY, 1995) e *Hybrid State Saving* (QUAGLIA; CORTELEESSA, 1997).

Nesta seção, será dado destaque a dois tipos de mecanismo em específico: o *Copy State Saving* e o *Sparse State Saving*. Esses mecanismos são particularmente importantes, pois estão atrelados à implementação desenvolvida neste trabalho.

2.2.3.1 Copy State Saving

O *Copy State Saving* caracteriza-se por salvar todos os estados de uma simulação distribuída, devendo a mesma retornar para o estado imediatamente mais

próximo e anterior ao LVT do evento que deve ser recuperado, quando da ocorrência de um erro de causa e efeito. Apesar de conceitualmente simples, este método apresenta as desvantagens de gastar muito tempo realizando o salvamento de estados e ocupar muito espaço de memória para armazenar todos os estados. Em um sistema que apresenta um baixo índice de *rollbacks*, este método pode ser ineficiente.

A figura 2.5 representa um processo P_1 de uma simulação distribuída em um diagrama espaço-tempo, que esteja utilizando o mecanismo *Copy State Saving*. a mensagem m possui o *timestamp* igual a 35 e, não obstante, é recebida pelo processo P_1 após ter executado o evento $e_{1,5}$, o que leva o LVT do processo para o valor 45, que é maior que o *timestamp* da mensagem m. Quando essa situação ocorre, pode-se dizer que a mensagem recebida é uma mensagem *straggler*, e esta ocasionará um *rollback* para o estado que contém o evento com o valor de LVT imediatamente menor ao *timestamp* da mensagem em questão. Como o mecanismo *Copy State Saving* salva todos os estados, a computação retorna para o evento $e_{1,4}$, com valor de LVT igual a 32, coloca o evento gerado pela mensagem m na fila de eventos futuros, e reprocessa os outros eventos a partir de $e_{1,5}$ com LVT igual a 35.

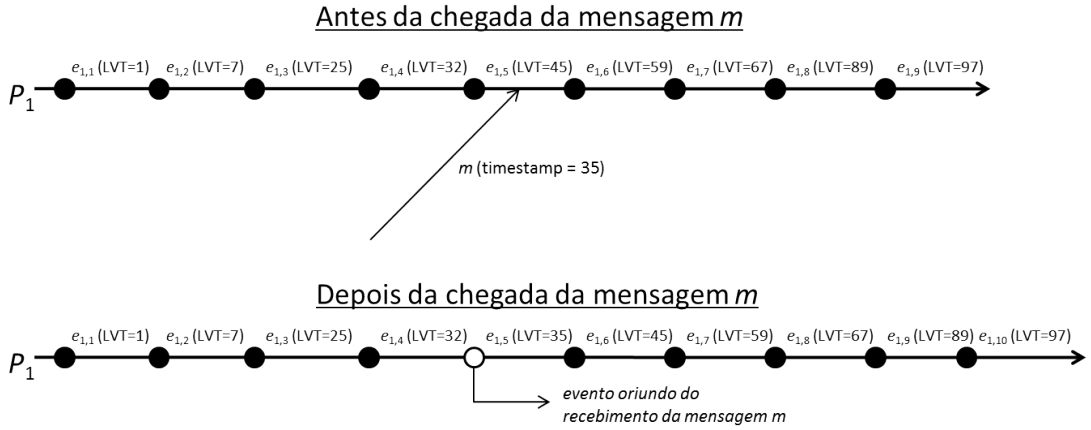

Figura 2.5: Mecanismo *Copy State Saving*

2.2.3.2 Sparse State Saving

Este mecanismo é um aprimoramento do *Copy State Saving*, de modo que não são todos os estados, gerados a cada evento processado, que são salvos. Essa otimização é pensada de forma que o mecanismo não perca tempo salvando estados que nunca serão utilizados. Neste mecanismo, os estados são salvos em um intervalo de iterações e não a cada evento executado. A vantagem principal deste mecanismo é a economia de memória, pois na ocorrência de um *rollback*, o estado recuperado será o estado salvo mais próximo do evento a ser processado, e não o imediato e anterior ao estado a ser recuperado.

Deste modo, os estados a serem salvos são definidos a partir da especificação de um intervalo que estabelece uma distância entre os eventos a serem salvos. Esse intervalo de gravação pode ser fixo (*Fixed Sparse State Saving*), ou variar (*Adaptive Sparse State Saving*) de acordo com a ocorrência de *rollbacks* na simulação (SKOLD; RONNGREN, 1996). Desta forma, considerando a segunda abordagem, quanto mais *rollbacks* ocorrerem, menor será o valor deste intervalo, de modo que o mecanismo *Sparse State Saving* (SSS) se assemelhará muito ao mecanismo *Copy State Saving*, sendo, entretanto, bastante eficiente se a incidência de *rollbacks* for menor, pois o tempo perdido no salvamento de estados que não serão utilizados será significativamente pequeno.

Na figura 2.6 os estados salvos pelo mecanismo SSS estão marcados, ao passo que os demais são descartados. Quando a mensagem *straggler* m é recebida pelo processo P_1, o mecanismo SSS realiza o *rollback* para o evento com LVT mais próximo e anterior aos eventos armazenados em sua memória. Neste caso, o evento $e_{1,2}$, com LVT igual a 7, adiciona o evento oriundo da mensagem m na fila de eventos futuros, e retorna o processamento a partir do próximo evento da fila de entrada, nesse caso $e_{1,3}$, com LVT igual a 25.

A região *coast forward* representa a distância entre o ponto de retorno ideal e o ponto de retorno real, ou seja, o mais próximo estado salvo pelo SSS. Quando um processo se encontra esta fase, nenhum evento de envio de mensagens é realizado. O tamanho desta região é um indicativo da eficiência do mecanismo, de modo que quando esta é muito grande isto indica um baixo desempenho, ao passo que o

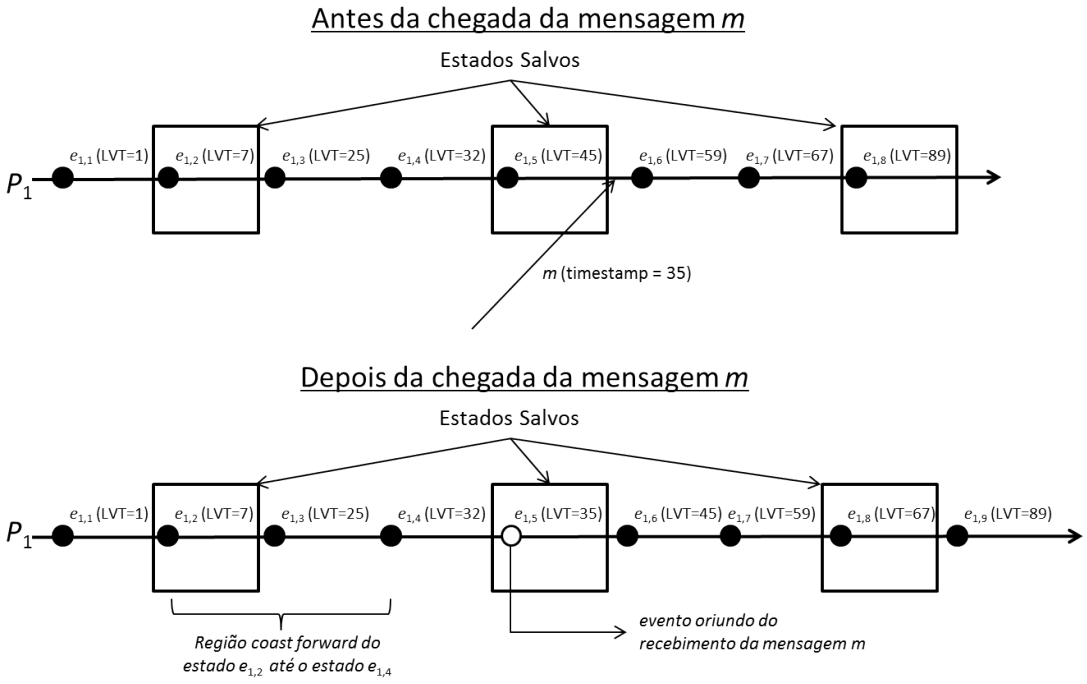

Figura 2.6: Mecanismo *Sparse State Saving*

oposto indica um bom desempenho.

2.2.4 Cálculo do GVT

Como especificado anteriormente, cada processo de uma simulação que executa o protocolo *Time Warp*, possui e utiliza três estruturas de dados essencias, para o funcionamento correto do mecanismo de recuperação de erros de causalidade: a fila de eventos futuros, a fila de antimensagens e a pilha de estados salvos. Conforme a simulação progride, o tamanho dessas estruturas pode crescer além das capacidades do sistema, causando uma situação de *overflow*, e consequentemente levando ao colapso do mesmo.

Para prevenir o problema de estouro da memória, é necessário um mecanismo de *garbage collection* (coletor de lixo) que descarte eventos, antimensagens e estados armazenados, que não serão mais utilizados em uma recuperação de sistema. O protocolo, então, utiliza um controle global da simulação, com base no cálculo

do GVT, para gerenciar a memória, garantindo a confiabilidade do sistema (JEFFERSON, 1985).

O GVT é calculado como o menor *timestamp* de todos os eventos, internos ou externos, ainda não processados, das filas, ou das mensagens em trânsito. Desta forma, é possível identificar a real posição lógica do sistema. Se o GVT equivale ao valor mínimo dos *timestamps* de todos os processos do sistema, pode-se concluir que, a partir daquele ponto, não haverá mais *rollbacks*. Entretanto, há uma limitação física que pode tornar o cálculo do GVT inconsistente, gerando um valor maior do que deveria gerar. Este problema ocorre devido ao atraso das mensagens durante o envio das mesmas na rede em que os computadores estão inseridos.

2.2.4.1 Mensagem Transiente

Mensagens transientes, ou mensagens em trânsito, são aquelas que sofreram atrasos durante o seu percurso e, não obstante, não constam na fila de eventos futuros quando o cálculo de GVT é efetuado. Deste modo, os valores de *timestamp* dessas mensagens não são levados em consideração no cálculo do GVT, o que pode levar a um valor incorreto do mesmo e provocar inconsistências na simulação.

Supondo que seja possível congelar a simulação em um dado instante de tempo T, e capturar o valor de LVT dos respectivos processos da simulação, o valor do GVT é o menor desses valores. Para a obtenção desses valores, é razoável considerar a utilização de um processo controlador C que receba, via mensagem, o valor de LVT de cada processo da simulação. Neste mecanismo, o processo controlador receberia, no início do procedimento de cálculo do GVT, os valores de LVT dos processos, calcularia o valor do GVT e, ao final do procedimento, reenviaria aos processos esse valor por meio de mensagens. Porém, esse cálculo pode ser incorreto caso uma mensagem transinte, enviada antes do início deste procedimento, possua um valor de *timestamp* menor que o GVT calculado.

Na figura 2.7 o processo controlador C envia mensagens aos demais processos da simulação para que registrem seus LVTs (40, 45 e 35) e, consequentemente, os enviem para C para o cálculo do GVT. Entretanto, a mensagem t do processo P_1, chega ao processo P_3 após o envio de seu valor de LVT para o processo controlador.

O valor do GVT calculado (35) é, pois, incorreto, uma vez que o *timestamp* da mensagem t (20) é menor.

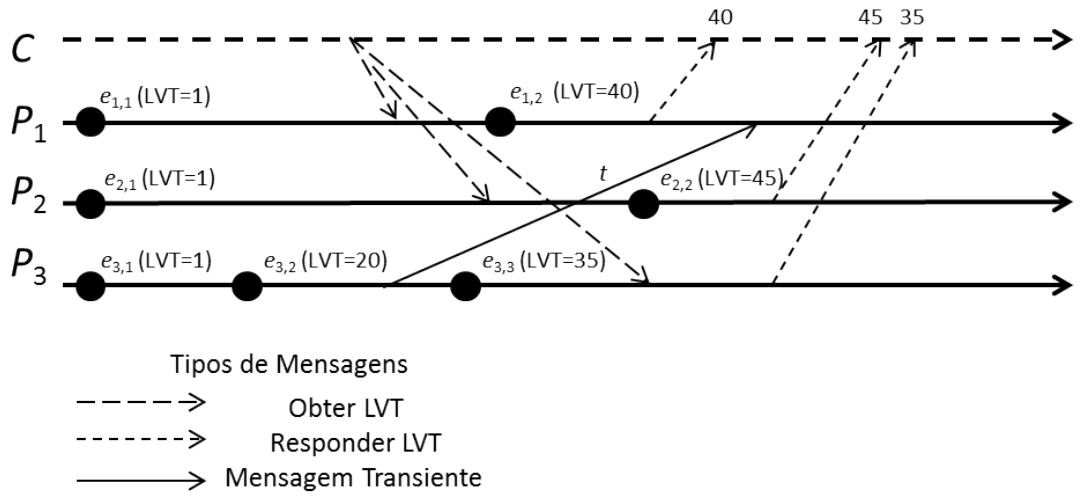

Figura 2.7: Uma mensagem transiente no sistema

Algumas abordagens foram propostas para ressolver esse problema da mensagem transiente. O algoritmo de Fujimoto (2000) resolve este problema através de um sistema de confirmação de recebimento de mensagens, que consiste, basicamente, em fazer com que todas as mensagens recebidas sejam confirmadas pelo processo receptor. Ao receber uma mensagem para calcular o GVT do processo coordenador, um processo interrompe seu procesamento, calcula seu LVT mínimo, e o envia ao processo coordenador.

O sincronismo inerente deste algoritmo, apesar de atender as especificações do sistema, limita o otimismo da simulação, degradando o seu desempenho, pois toda a computação é interrompida durante o cálculo do GVT. Entretanto, a não interrupção do processamento pode provocar outro problema: o relatório simultâneo. Este problema, basicamente, consiste na não inclusão dos tempos lógicos das mensagens transientes no cálculo do GVT, provocando erros.

A fim de resolver esse problema do relatório simultâneo, e da degradação de desempenho, causada pelo bloqueio da computação do sistema, Samadi (1985) e Mattern (1993) propuseram diferentes mecanismos, que resolvem os problemas da mensagem transiente e do relatório simultâneo de modo assíncrono, ou seja,

sem a necessidade de interromper a simulação durante o cálculo do GVT. Este último, utilizado neste trabalho para o cálculo do GVT, será explicado na subseção seguinte.

2.2.4.2 Algoritmo de Mattern

O algoritmo de Mattern (1993) possibilita que o GVT seja calculado sem a necessidade de sincronização global, e sem a necessidade da troca de mensagens de confirmação entre os processos da simulação, como no algoritmo de Samadi (1985). Este método utiliza o conceito de corte consistente que, basicamente, divide a computação em duas épocas: "passado" e "futuro", para garantir que nenhuma mensagem deixe de ser computada durante a obtenção do GVT do sistema. Um corte consistente é constituido por um subconjunto de eventos salvos, de modo que, nenhum evento que antecede o corte seja precedido por um evento que sucede o corte (ZIMMERMAN; KNOKE; HOMMEL, 2006).

O funcionamento deste algoritmo consiste em utilizar dois cortes, de modo que a simulação seja dividida em duas áreas, "branca" e "vermelha", que se sucedem no momento do cálculo do GVT. As áreas branca e vermelha, respectivamente, representam a execução normal da simulação e o momento que o GVT está sendo calculado. Todos os processos iniciam na área branca, no primeiro corte são marcados com a cor vermelha e retornam a cor branca após o segundo corte. A área na qual um determinado processo se encontra, é indicada por meio de um *flag*, cujo valor indica se este está na área branca ou vermelha. Consequentemente, as mensagens enviadas quando o *flag* de um processo sinalizar estar em um área branca, podem ser chamadas de "mensagens brancas". Analogamente, as mensagens enviadas quando o processo estiver com o *flag* sinalizando estar na área vermelha, podem ser chamadas de "mensagens vermelhas".

A função do primeiro corte é delimitar a área vermelha à esquerda. O motivo de existência desta área é assegurar que todas as mensagens transientes brancas sejam recebidas dentro dela, sem atravessar o segundo corte, que delimita a área vermelha à direita. A idéia é impedir que as mensagens enviadas antes do primeiro corte, atravessem o segundo corte. Essa restrição possibilita o cálculo correto do GVT

dentro da área vermelha, de modo que todas as mensagens transientes, anteriores ao primeiro corte, chegarão antes do GVT ser calculado, impedindo que situações como a da figura 2.7 ocorram.

No caso deste algoritmo, o cálculo do GVT é assíncrono, sendo pois, desnecessário que os cortes construídos sejam consistentes. As mensagens que criariam um corte inconsistente podem ser ignoraradas, pois elas devem ter um *timestamp* maior que o GVT calculado usando um corte consistente. Desta forma, o cálculo do GVT deve ser o mínimo entre o menor LVT de um evento não processado por cada processo no seu ponto de corte e o menor *timestamp* de qualquer mensagen transiente vermelha que atravesse o segundo corte do "passado" para o "futuro", ou seja, da área vermelha para a branca.

A figura 2.8 ilustra o funcionamento deste algoritmo, os cortes e as áreas branca e vermelha. O algoritmo determina as mensagens transientes utilizando dois cortes, $C1$ e $C2$, e calcula o GVT na fronteira do corte $C2$. No primeiro corte, $C1$, é solicitado aos processos que iniciem o registro do menor *timestamp* das mensagens que enviarem, independente se estas irão atravessar o segundo corte $C2$ (transientes) ou não. Todas as mensagens vermelhas devem ser consideradas no cálculo do GVT.

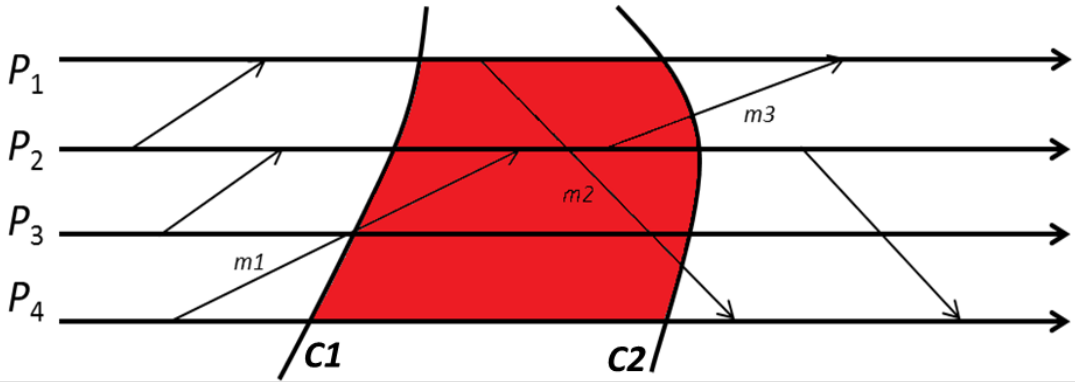

Figura 2.8: Funcionamento do algoritmo de Mattern

No exemplo da figura 2.8, as mensagens vermelhas são $m2$ e $m3$, e seus *timestamps* são considerados no cálculo do GVT, ao passo que todas as mensagens brancas enviadas antes do corte $C1$, como $m1$, devem ser recebidas dentro da área

vermelha, ou seja, antes do corte $C2$. O segundo corte, então, só será obtido após o recebimento da mensagem $m1$.

Uma vez que cada processo da simulação registre, ao menos, as quantidades de mensagens brancas que envia e recebe, é possível determinar, a partir do corte $C1$, quantas mensagens brancas faltam ser recebidas, em cada um desses processos. Esse cálculo pode ser feito, por cada processo, subtraindo-se a quantidade de mensagens recebidas da quantidade de mensagens enviadas por outros processos para o processo em questão. Para tanto, é necessário que esses dados sejam reunidos de modo que o cálculo possa ser efetuado. Isso pode ser realizado tanto por intermédio de um processo observador, quanto utilizando a abordagem de eleição em anel, passando o controle de processo a processo, até que os cortes estejam efetuados (MATTERN, 1993 apud MOREIRA, 2005). A abordagem utilizada para o algoritmo de Mattern neste trabalho é a primeira, e será melhor detalhada no capítulo 4.

O GVT, então, é calculado, no momento do segundo corte, como o menor valor entre:

- Os *timestamps* das mensagens vermelhas, mesmo as que estiverem em trânsito, enviadas por todos os processos;
- Os LVTs de todos os processos da simulação.

2.3 Rollback Solidário

De forma análoga ao protocolo *Time Warp*, o protocolo proposto por Moreira (2005) também adota a abordagem otimista e se utiliza de *rollbacks* para restaurar a simulação após a ocorrência de erros de causa e efeito. Entretanto, embora ambos os protocolos sejam otimistas, há mudanças estruturais significativas que fazem o *Rollback* Solidário ser um protocolo diferente do *Time Warp*, e não o transformam em uma variante do *Time Warp*.

A diferença principal entre os dois protocolos se dá na forma de tratar as mensagens *stragglers* recebidas pelos processos da simulação e, consequentemente,

na forma como realizam o procedimento de *rollback*. Diferentemente do protocolo de Jefferson (1985), o *Rollback* Solidário não necessita de antimensagens durante a recuperação do sistema, e para isto, utiliza o conceito de *checkpoints* globais consistentes (MOREIRA; SANTANA; SANTANA, 2005).

Complementarmente à diferença no procedimento de *rollback*, os dois protocolos possuem diferenças que, basicamente, gravitam em torno do seguinte eixo: (1) o procedimento utilizado quando uma mensagem *straggler* surge no sistema; (2) a forma de organizar os *checkpoints* através do conjunto de linhas de recuperação e (3) a maneira como é determinado o estado para onde os processos deverão retornar, na ocorrência de um *rollback*. Nas seções seguintes serão apresentados os aspectos gerais do protocolo *Rollback* Solidário bem como seu funcionamento.

2.3.1 Aspectos Gerais

Quando uma mensagem *straggler* é identificada em uma sistema que está executando o *Rollback* Solidário, o protocolo identifica o ponto de retorno para todos os processos que devem desfazer a computação errônea, e assim, todos estes iniciam um procedimento de *rollback* simultaneamente. O ponto de recuperação em que os processos devem realizar a recuperação do sistema, é identificado por um mecanismo que se utiliza da teoria dos *checkpoints* globais consistentes. O nome "*Rollback* Solidário" se deve justamente ao fato de que para cada operação de *rollback*, desencadeado por mensagem *straggler*, o protocolo se baseia em dados coletados por todos os processos da simulação, ou seja, o procedimento de *rollback* é "solidário", e não ocorre de forma isoladada e assíncrona como no *Time Warp*.

O uso de um mecanismo que assegure *checkpoints* globais consistentes elimina a necessidade do uso de antimensagens para a restauração do sistema em caso de *rollback*. Isso é vantajoso em termos de ganho de economia de memória, pois não não existe a necessidade de armazenar listas de cópias das mensagens enviadas e de antimensagens. A eliminação do uso de antimensagens também traz ganhos em termos de comunicação. Por não haver antimensagens ocupando os canais de comunicação do sistema, o tráfego na rede é diminuído, pois, quando ocorre uma operação de *rollback*, o ponto de retorno de cada processo é enviado via *broadcast*

pelo processo observador (MOREIRA, 2005).

Devido aos *rollbacks* deste protocolo serem solidários, não há *rollbacks* em cascata pois não existe a possibilidade de um *rollback* primário gerar outro *rollback* secundário, e assim sucessivamente. O uso de *checkpoints* globais consistentes retorna a computação para um conjunto de estados que não possuem relação de causalidade entre si, além de facilitar o cálculo do GVT.

Em relação ao mecanismo de salvamento de estados, há duas formas distintas de realizá-lo: em intervalos de tempo estipulados, similarmente ao *Sparse State Saving* utilizado pelo *Time Warp*, ou em situações específicas, nas quais é necessário provocar *checkpoints* forçados para se garantir a consistência do protocolo, afim de assegurar que não hajam *checkpoints* locais inúteis, ou seja, cada um deles deve pertencer a pelo menos um *checkpoint* global consistente (CGC) (MANIVANNAN; SINGHAL, 1999).

2.3.2 Tratamento de Mensagens

O *Rollback* Solidário trata as mensagens inerentes ao modelo simulado de forma análoga ao *Time Warp*. Se o *timestamp* da mensagem for maior ou igual ao LVT do processo que a recebeu, esta é inserida na fila de eventos futuros e processada normalmente com o andamento da simulação. Caso contrário, ela é tratada como uma mensagem *straggler* desencadeando uma operação de *rollback*. A diferença, neste caso, se dá na forma como o procedimento de *rollback* é realizado no *Rollback* Solidário. As seções seguintes esclarecem como esse mecanismo funciona.

O diagrama da figura 2.9 ilustra os passos executados por um processo para o recebimento de uma mensagem inerente ao modelo simulado. Inicialmente, é verificado o tipo da mensagem, ou seja, se ela é *staggler* ou não. Caso a mensagem recebida seja "normal", isto é, se não viola a relação de causalidade, ela apenas é inserida na fila de eventos futuros. Caso contrário, o sistema localiza um CGC que contenha o estado para o qual o processo deve retornar, sincronizando os demais processos para continuar a partir do estado recuperado para cada um deles.

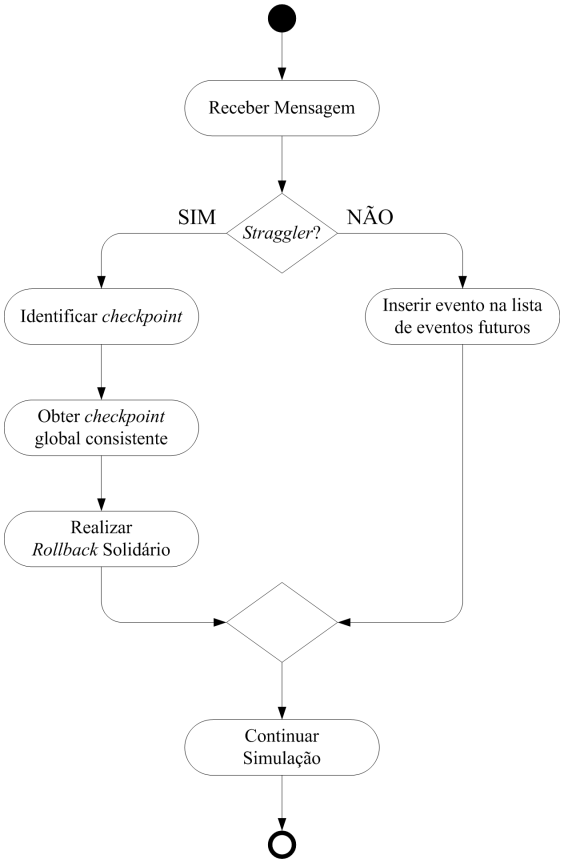

Figura 2.9: Tratamento de Mensagens no *Rollback* Solidário (MOREIRA, 2005)

2.3.3 Linhas de Recuperação

Um *checkpoint* global consistente constitui-se do conjunto de vários *checkpoints* locais, sendo um para cada processo. Quando um *rollback* é efetuado no sistema, o protocolo *Rollback* Solidário identifica um CGC para que a simulação possa retomar seu processamento. Qualquer *checkpoint* global identificado pelo sistema serviria, entretanto, por questões de desempenho, é desejável que se escolha o *checkpoint* mais recente afim de minimizar o volume de processamento perdido, reduzindo assim o tamanho da região de *coast foward* a ser reprocessada. Esse *checkpoint* escolhido para se fazer a restauração é chamado de linha de recuperação (*recovery line*). (RANDELL; CHIU; YOUNG, 1975, 1996 apud MOREIRA, 2005)

No decorrer da simulação, o sistema irá possuir um conjunto de linhas de

recuperação que serão utilizadas, caso necessário, para recuperar a simulação, retornando-a para um estado consistente. Todas estas linhas de recuperação possibilitam ao sistema retornar para um estado global consistente, o que as diferencia, portanto, está no custo da recuperação para a simulação.

A figura 2.10 ilustra um diagrama espaço-tempo executando uma simulação com três processos: P_1, P_2 e P_3. Nele destacam-se três linhas de recuperação: A, B e C, representadas pelas linhas sinuosas e pontilhadas. Quando a mensagem *straggler* é recebida pelo processo P_1, o sistema pode optar por retornar para qualquer uma dessas linhas. Entretanto, a linha de recuperação C representa um retrocesso menor, logo, apresenta um menor custo para a simulação.

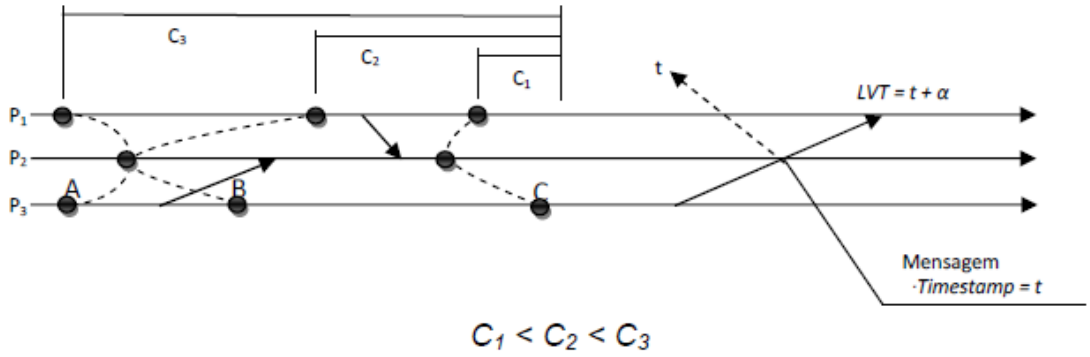

Figura 2.10: Exemplos de linhas de recuperação (CRUZ, 2009)

O custo para o retorno da simulação pode ser definido como o intervalo entre o ponto ao qual o processo deveria retornar e a linha de recuperação utilizada para a recuperação do sistema. Ao se comparar estes intervalos, pode-se calcular o prejuízo provocado por cada linha de recuperação apresentada, sendo $C1$, $C2$ e $C3$ os custos para as linhas C, B e A, respectivamente. Com base nessa comparação, pode-se atribuir às linhas C e A o que Moreira (2005) define como linhas de recuperação em máxima e mínima, respectivamente. A linha de recuperação máxima (neste caso C) é a que traz menor prejuizo, e não obstante, um menor custo de recuperação ao sistema. Diamentralmente oposta, a linha de recuperação mínima (neste caso A) é a que traz maior prejuizo, e não obstante, um maior custo de recuperação ao sistema.

A idéia central do mecanismo de *rollback* deste protocolo é sempre retornar para linha de recuperação máxima obtida. Entretando, há situações em que não será possível retornar para a última linha de recuperação obtida, pois o *timestamp* da mensagem *straggler* que o gerou pode requerer que a simulação retorne para linhas anteriores. O mecanismo para a obtenção dessas linhas de recuperação, e para controle de consistência das mesmas, será explicado na próxima seção.

2.3.4 Mecanismo para Obtenção de Checkpoints

Duas abordagens para obtenção das linhas de recuperação, ou *checkpoints* globais consistentes, podem ser utilizadas pelo protocolo *Rollback* Solidário: o mecanismo síncrono e o mecanismo semi-síncrono. Cada um desses mecanismos impacta no desempenho da simulação de formas diferentes, de modo que possuem uma vantagem e desvantagem quanto ao seu uso. O mecanismo síncrono possui a vantagem de otimizar o uso da memória do sistema, entretanto, acarreta como desvantagem o tempo perdido durante a obtenção das linhas de recuperação. Em contrapartida, o mecanismo semi-síncrono possui a vantagem de não interromper o sistema durante a obtenção das linhas de recuperação, ao passo que acarreta, com a desvantagem, a utilização de uma quantidade maior de memória de armazenamento.

Independente da abordagem utilizada, o mecanismo de obtenção de *checkpoints* requer que sejam coletadas informações de todos os processos da simulação. A forma mais evidente de tornar esse mecanismo possível, é dispor de um processo controlador que solicite essas informações para os demais processos. Moreira (2005) apresenta uma maneira alternativa de fazer essa coleta sem a utilização de um processo observador. No entanto, será focado apenas no funcionamento desse mecanismo por intermédio de um processo observador, visto que a implementação utilizada neste trabalho utiliza-se deste paradigma, e confere outras atribuições à esse processo a serem detalhadas melhor no capítulo 4.

Ambas as abordagens são confiáveis, e permitem a obtenção de *checkpoints* globais consistentes que possam ser usados em um procedimento de *rollback*, para a recuperação de erros de causa e efeito. Na abordagem síncrona a simulação é interrompida para a obtenção das linhas de recuperação, ao passo que isso não é

necessário na abordagem semi-síncrona.

2.3.5 Abordagem Semi-Síncrona para Obtenção de Checkpoints

Nesta abordagem, o processo coordenador assume uma função mais "passiva" durante o cálculo das linhas de recuperação, apesar de atuar ativamente quando da execução de um procedimento de *rollback* pelo sistema. O processo coordenador assume, portanto, a função de um processo observador no método semi-síncrono.

Para que se possa compreender a abordagem semi-síncrona para a obtenção de linhas de recuperação, é necessário abordar, primeiramente, sobre como o mecanismo de salvamento de estados opera. Este mecanismo deve garantir que todo *checkpoint* local faça parte de pelo menos um *checkpoint* global consistente, de modo que isso possibilite uma melhora de desempenho da simulação, evitando desperdício de memória e tempo de execução.

2.3.5.1 Mecanismo de Salvamento de Estados

Há dois tipos possíveis de *checkpoints* locais no protocolo *Rollback* Solidário: básicos e forçados. Os primeiros emanam da própria ocorrência de eventos da simulação, enquanto os segundos são realizados de modo a fazer com que cada *checkpoint* local pertença a pelo menos um *checkpoint* global consistente, isto é, a pelo menos uma linha de recuperação do sistema.

O mecanismo de salvamento dos *checkpoints* básicos é semelhante ao *Sparse State Saving*, mecanismo usado pelo *Time Warp*, no qual os estados salvos são escalonados dentro de intervalos de tempo, que podem ser fixos ou variáveis. No caso de intervalos variáveis, a ocorrência de *rollbacks* pode ser usada como critério inversamente proporcional ao tamanho deste intervalo.

O salvamento de *checkpoints* forçados, em contrapartida, tem como base o algoritmo *Fixed Dependency After Send* (FDAS), proposto por WANG (1997), e que implementa o padrão *No Receive After Send* (NRAS). Este padrão garante a não ocorrência de *checkpoint* inúteis, isto é, aqueles que não pertencem a nenhum *checkpoint* global consistente (MANIVANNAN; SINGHAL, 1999), através da elimina-

ção de todos os *z-paths* (caminhos-Z) não causais, entre dois *checkpoints* quaisquer, não necessariamente distintos (quando são idênticos, o *z-path* em questão é um caso particular denominado *z-cycle*). Caminhos-Z, *z-paths*, ou ainda, *zigzag paths* podem ser definidos como uma sequência de mensagens $M = \{m_1, m_2, \ldots, m_n\}$ $(n \geq 1)$ entre dois *checkpoints* Ck_i e Cl_j tal que: (1) m_1 é enviada por p_i após Ck_i; (2) se m_a $(1 \leq a < n)$ é recebida por p_r, então $m_a + 1$ é enviada por p_r neste mesmo intervalo de *checkpoints* ou em algum intervalo posterior a este e (3) m_p é recebida por p_j antes de Cl_j (NETZER; XU, 1995).

A figura 2.11 ilustra um exemplo de um sistema com um *zigzag path*. Nesta ilustração, os *checkpoints* C_1^1 e C_3^1 são concorrentes entre si, entretanto, não existe nenhum *checkpoint* no processo P_2 que possa se unir a estes *checkpoints* locais para formar um *checkpoint* global consistente.

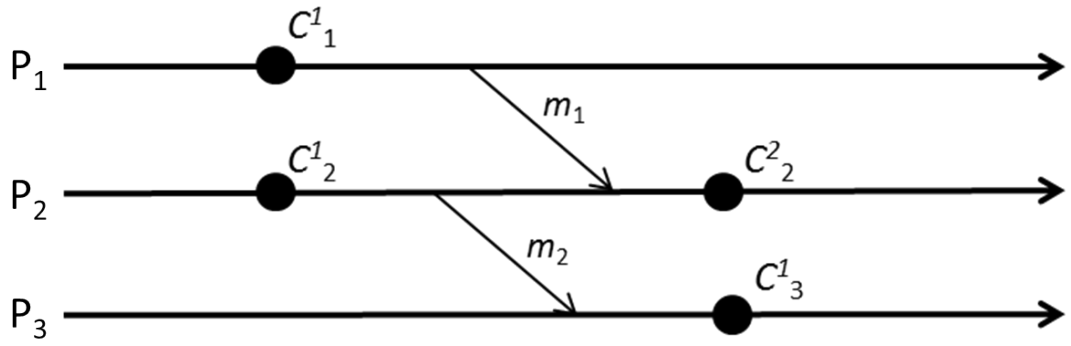

Figura 2.11: Exemplo de um sistema com z-path

Então, consequentemente, para se assegurar que os *checkpoints* locais da simulação pertençam a um CGC, estes devem possuir o padrão ZPF (*Z-path Free*), que consiste na eliminação dos *sibling causal path*, ou seja, *z-paths* não causais correspondentes àqueles que não possuam caminhos duplicados causalmente (MANIVANNAN; SINGHAL, 1999). O FDAS, algoritmo usado por esse mecanismo, assegura esse padrão. O diagrama espaço-tempo apresentado na figura 2.12 representa um padrão de checkpoints ZPF. Nesta figura, os *checkpoints* forçados, representados pelos círculos brancos, previnem caminhos-Z não causais que não possuem *sibiling causal path* correspondentes.

Além do padrão NRAS, o algoritmo FDAS satisfaz a propriedade *Rollback-*

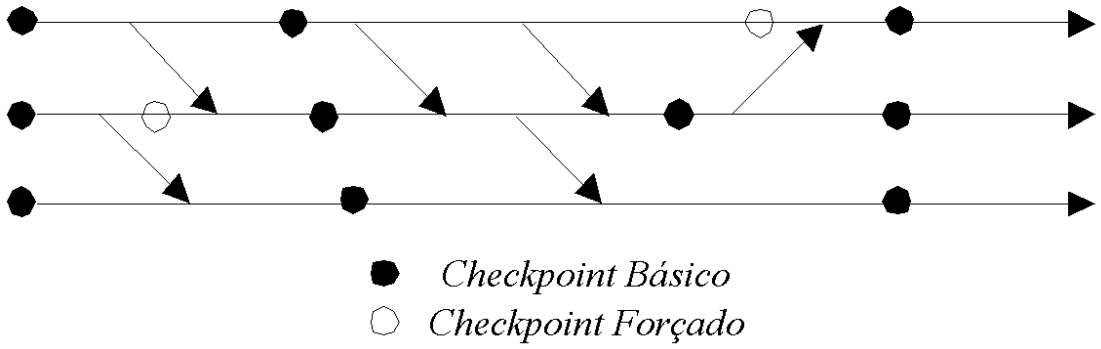

Figura 2.12: Padrão de checkpoints ZPF (MOREIRA, 2005)

Dependency Trackability (RDT) (WANG, 1997). Esta propriedade possibilita ao protocolo rastrear todas as dependências entre *checkpoints* em tempo de execução, por meio do uso de vetores de dependências. A utilização desses vetores permite ao protocolo armazenar os *checkpoints* globais consistentes de forma a facilitar a identificação de um *checkpoint* específico, durante algum procedimento de *rollback*.

2.3.5.2 Vetores de Dependências

Como consequência da propriedade RDT, todos os processos possuem vetores de dependências, conhecidos também como relógios vetoriais, os quais possuem a função de identificar a relação de precedência causal entre os *checkpoints* do sistema. Esses vetores são de ordem n, na qual n é igual ao número de processos da simulação, sendo que cada posição do vetor representa um processo do sistema. Ao início da simulação, cada processo executa um *checkpoint*, de modo que a posição i do vetor do processo P_i é incrementado em uma posição. Desta forma, em uma simulação com quatro processos (n igual a 4), por exemplo, o relógio lógico do processo P_1, após o primeiro *checkpoint*, ficaria igual a $VD_1 = \{1, 0, 0, 0\}$. De maneira análoga, os vetores de P_2, P_3, P_4 ficariam, respectivamente, igual a $VD_2 = \{0, 1, 0, 0\}$, $VD_3 = \{0, 0, 1, 0\}$ e $VD_4 = \{0, 0, 0, 1\}$.

A estrutura de mensagens do protocolo *Rollback* Solidário permite aos processos da simulação enviarem os seus vetores de dependências a cada mensagem para outros processos. Isso faz com que os processos receptores atualizem seus

vetores de dependências, de acordo com o valor dos relógios vetoriais recebidos. Essa atualização ocorre da seguinte forma:

1. O processo receptor compara os valores do seu próprio vetor com as posições do vetor de dependências recebido. O maior valor entre os vetores é usado na atualização do relógio lógico atual.

2. O processo realizará *checkpoint* forçado, caso tenha recebido a informação de um *checkpoint* novo e enviado uma mensagem após o seu *checkpoint* local mais recente.

Tomando como base o exemplo anterior com quatro processos, se o processo P_3 envia seu vetor na posição $VD_3 = \{0, 0, 4, 0\}$, por meio de mensagem, para o processo P_1, com $VD_1 = \{5, 0, 0, 0\}$, então o valor do relógio lógico de P_1, caso o sistema tenha realizado um *chekpoint* forçado, será atualizado como $VD_1 = \{6, 0, 4, 0\}$. Caso contrário, como $VD_1 = \{5, 0, 4, 0\}$.

As mensagens trocadas entre os processos, bem como as mensagens enviadas ao processo observador para o cálculo das linhas de recuperação, carregam também os respectivos LVTs de cada *checkpoint* local armazenado. O vetor de LVTs é utilizado pelos processos receptores durante a análise da eliminação de mensagens na fila de eventos futuros durante um procedimento de *rollback*.

2.3.5.3 Identificação das Linhas de Recuperação

Os *checkpoints* locais, básicos ou forçados, precisam ter seus vetores de dependências computados no processo observador, de modo que possa se calcular as linhas de recuperação necessárias ao mecanismo de *rollback* do protocolo. Para tanto, o processo observador dispõe de um relógio matricial de ordem n, com n igual ao número de processos da simulação. Nesta matriz quadrada, cada linha i representa o último relógio vetorial do processo P_i, enviado ao processo observador. Cada coluna j representa a relação de dependência causal dos processos da simulação com o processo P_j. Desta forma, a diagonal da matriz representa a linha

de recuperação a qual o sistema deve retornar, caso aconteça um procedimento de *rollback* no sistema.

Ao início da execução, a matriz de *checkpoints* do processo observador é similar a matriz identidade. Sua diagonal com elementos com valor igual a 1, pois os processos ao início de sua execução realizam um *checkpoint* inicial. Os demais elementos, portanto, tem valor igual a 0. Uma simulação com quatro processos teria, ao início da simulação, formato igual a:

$$M_{4x4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

O processo observador não interrompe a execução da simulação para a obtenção de CGCs na abordagem semi-síncrona. A consistência e validade das linhas de recuperação, obtidas através da diagonal da matriz, é aferida comparando-se cada valor da diagonal da matriz M com os demais elementos das colunas ao qual esse valor pertence. Cada elemento M_{ij}, tal que $i = j$, deve ser maior que os demais elementos da coluna j ao qual pertence. Na matriz a seguir, a diagonal $d = \{3, 4, 4, 2\}$ representa um CGC e, portanto, uma linha de recuperação válida, podendo ser usada na recuperação do sistema quando um *rollback* for desencadeado.

$$M_{4x4} = \begin{pmatrix} 3 & 0 & 1 & 1 \\ 1 & 4 & 2 & 0 \\ 0 & 3 & 4 & 1 \\ 2 & 2 & 3 & 2 \end{pmatrix}$$

A matriz de *checkpoints* é atualizada de acordo com o andamento da simulação. O diagrama espaço-tempo da figura 2.13 ilustra uma simulação com quatro processos, P_1, P_2, P_3 e P_4 e com um processo observador O, em que os *checkpoints* locais são enviados ao processo observador, de modo que as linhas de recuperação possam ser determinadas. As linhas tracejadas representam mensagens, ineren-

tes à simulação, de um processo ao outro, ao passo que os círculos representam *checkpoints* locais. O processo observador determina as linhas de recuperação, $\{1,1,1,1\}$, $\{2,1,1,1\}$, $\{3,2,2,2\}$, $\{3,3,3,2\}$ e $\{4,4,4,3\}$: as diagonais da matriz a cada atualização.

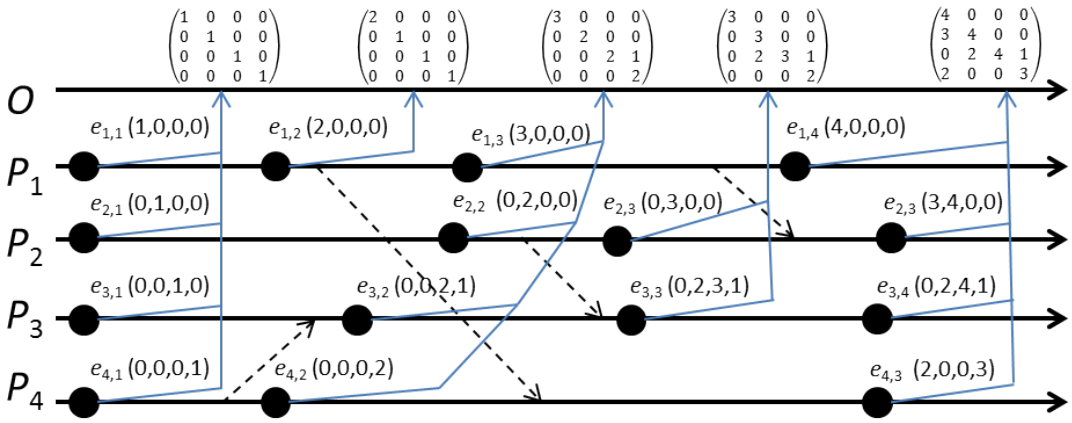

Figura 2.13: Registro de *checkpoints* do protocolo *Rollback* Solidário na abordagem semi-síncrona

As linhas de recuperação geradas precisam ser armazenadas para uso posterior, no caso de uma recuperação de erros de causalidade. O uso de uma matriz quadrada pelo mecanismo semi-síncrono é bastante simples e de fácil implementação, entretanto, a solução baseada em matriz não consegue identificar todos as linhas de recuperação possíveis (MOREIRA, 2005). Uma solução alternativa é fazer o uso de listas encadeadas para armazenar os *checkpoints* locais. Essa técnica apesar de identificar um número maior de CGCs, e possibilitar o retorno da computação para uma linha de recuperação que cause menos prejuízo ao sistema, apresenta a desvantagem de demorar um período de tempo maior para localizar o melhor CGC para a recuperação do sistema.

O processo observador deve escolher a linha de recuperação mais adequada para a recuperação do sistema. Essa escolha deve se basear não apenas na linha que cause menos prejuízo ao sistema, mas na também a linha que satisfaça a necessidade de recuperação do sistema, e retorne a computação para o *checkpoint* imediatamente anterior ao valor do relógio lógico da mensagem *straggler* que desencadeou o *rollback*. Na figura 2.14, as mensagens *stragglers*, m_1, m_2 e m_3, fazem

a computação retornar, respectivamente, para as linhas de recuperação: C_2, C_4 e C_5. Neste caso, um retorno para as linhas C_2 e C_4 causaria menor prejuízo para simulação do que C_1 e C_3, respectivamente, no caso dos *rollbacks* causados pelas mensagens m_1 e m_2.

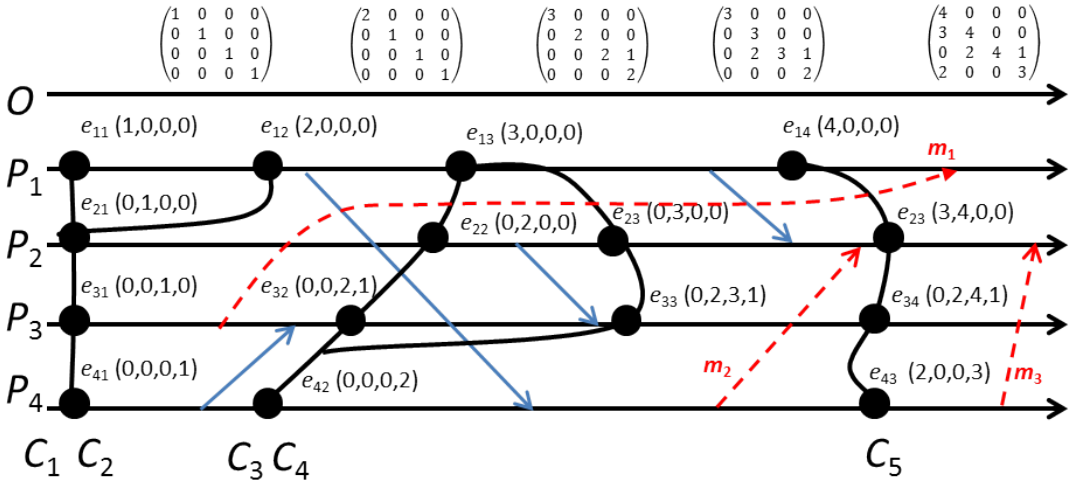

Figura 2.14: Linhas de recuperação do sistema

2.3.5.4 Tratamento de Rollbacks

Assim como na abordagem síncrona, o mecanismo para tratamento de *rollbacks*, no protocolo *Rollback* Solidário, envolve participação ativa do processo observador, a partir do momento que este recebe a mensagem para iniciar o procedimento de *rollback* no sistema.

Quando uma mensagem *straggler* chega em qualquer processo da simulação, os seguintes passos são executados na sequência:

1. O processo que recebeu a mensagem *straggler* envia uma mensagem **RealizaRollback** para o processo observador.

2. O processo observador, ao receber a mensagem **RealizaRollback**, identifica a linha de recuperação máxima, ao qual o sistema deve retornar, e envia mensagens **CheckpointDeRetorno** para todos os processos. Essas mensagens contêm os pontos ao quais o sistema deve retornar, para cada processo.

3. Ao receberem as mensagens **CheckpointDeRetorno** do processo observador, com seus respectivos pontos de retorno, os processos da simulação realizam *rollback* para o ponto especificado.

4. Após cada processo efetuar seu retorno ao *checkpoint* especificado, o procedimento de *rollback* está finalizado.

Os processos da simulação ficam congelados durante o período de tempo em que recebem o ponto de retorno do processo, e completam o retrocesso de suas computações.

Essa abordagem de tratamento de *rollback*, apesar de simples, acarreta alguns problemas, inerentes ao funcionamento do protocolo, que requerem algumas modificações no método apresentado. Essas alterações serão apresentadas nas próximas seções.

2.3.6 Problemas do Rollback Solidário

O *Rollback* Solidário possui, também, limitações que emanam diretamente de suas características. Mais especificamente, quatro delas se destacam, embora uma delas surja como consequência de um dos outros três restantes:

1. Anomalia do Retorno Desnecessário: o ponto de retorno recebido pelo processo observador, muitas vezes pode significar um retrocesso desnecessário, se o estado atual do processo não possuir nenhuma relação de dependência causal com o processo que disparou o *rollback*;

2. *Livelock*: decorrente do problema da Anomalia do Retorno Desnecessário, é uma situação na qual a simulação entraria em um ciclo de *rollbacks* e, desta forma, não progrediria mais. Isso é possível, pois se o sistema é restaurado para a última linha de recuperação identificada pelo processo observador, o evento que gerou o *rollback* pode ter sido desfeito e poderá ser gerado novamente após a simulação ter retomado a sua execução.

3. *Rollbacks* Simultâneos: dois ou mais procedimentos de *rollback* podem acontecer ao mesmo tempo. Quando essa situação ocorre, podem ser geradas inconsistências no sistema devido à perda de informações, dependendo da ordem em que esses *rollbacks* são tratados pelo protocolo.

4. Mensagem transiente: se um processo envia mensagem a outro, pode ocorrer atrasos na rede. Isso pode fazer com que essa mensagem chegue após um procedimento de *rollback* que restaurou a simulação para uma linha de recuperação anterior ao envio desta mensagem. Desta forma, o sistema estaria processando uma mensagem a qual deveria ter eliminado e, desta forma, processando a mesma mensagem mais de uma vez.

Esses problemas podem ser corrigidos com alterações no modo de tratamento do procedimento de *rollback*, em especial com a adição de informações na estrutura de mensagens. O detalhamento dessas soluções é discutido em Moreira (2005).

2.4 Biblioteca de Comunicação MPI

O particionamento dos modelos matemáticos em vários processos, para posterior execução de uma simulação distribuída, pode ser feito utilizando duas abordagens diferentes: SRIP (*Single Replication in Parallel*) e MRIP (*Multiple Replication in Parallel*) (YAU, 1999). A abordagem SRIP, utilizada no projeto desenvolvido neste trabalho, consiste basicamente em mapear um modelo de simulação e dividi-lo em várias partes, as quais podem ser executadas de forma paralela em um ambiente distribuído. A abordagem MRIP, de modo diferente, consiste em replicar um mesmo programa de simulação, sem particionamento do modelo, em vários processadores.

Em uma simulação distribuída, os processos precisam se comunicar, e para isso precisam de uma padronização das mensagens para estabelecer essa comunicação. Nesse sentido, o MPI é uma especificação de interface para uma biblioteca de passagem de mensagem (MPI-FORUM, 2015) que determina como os processos se comunicam. Por tratar-se de uma biblioteca de comunicação inter-processual, o

MPI dispõe de funções para envio e recebimento de mensagens, além de outras responsáveis por implementar as funcionalidades específicas do MPI.

A próxima seção versa a respeito das características e funcionalidades do MPI.

2.4.1 Características

Nessa biblioteca, inicialmente, um conjunto fixo de processos é criado, embora estes possam executar diferentes programas. Uma das características do MPI é assegurar a ordem (FIFO) do canal entre dois processos, de modo que se um processo envia duas mensagens, na sequência, para outro, o processo receptor as receberá na mesma ordem em que foram enviadas (MPI-FORUM, 2015). Isto traz facilidades na implementação dos mecanismos de tratamento de mensagens em ambos os protocolos apresentados anteriormente.

O MPI possui recursos importantes, tais como: comunicação ponto a ponto, operações coletivas, grupo de processos, operações coletivas extendidas, domínios de comunicação, contextos, topologias de processos, comunicadores, rotinas de ambiente, entre outros (MPI-FORUM, 2015).

2.4.2 Comunicação Ponto a Ponto

O conceito de comunicação ponto a ponto refere-se a um padrão de troca de mensagens, por meio de funções de envio e recebimento, entre um par de processos. Esse mecanismo de comunicação básico embasa a maior parte da construcão do MPI. Existem dois esquemas de comunicacão suportadas pela biblioteca MPI: a comunicação bloqueante e a não-bloqueante, ou, respectivamente, comunicação síncrona e assíncrona (COULOURIS; KINDBERG; DOLLIMORE, 2013).

A comunicação síncrona ocorre quando as operações de envio e recebimento de dados bloqueiam ou suspendem a execução do processo até que comunicacão esteja concluída. Neste caso, é necessário uma confirmação do recebimento da mensagem por parte do receptor, para que o emissor seja desbloqueado.

Em contrapartida, a comunicação assíncrona ocorre quando as operações de

envio e recebimento de dados não bloqueiam ou suspendem a execução do processo até que comunicacão esteja concluída. Neste caso, não é necessário a confirmação do recebimento de mensagens por parte do receptor. O emissor das mensagens pode continuar executando seu processamento sem interrupção. O receptor, por sua vez, não permanecerá bloqueado caso a mensagem, enviada a ele, não tenha chegado.

As operações de envio e recebimento de mensagens em ambas as abordagens, síncrona ou assíncrona, são implementadas por rotinas de envio e recebimento de mensagens, bloqueantes e não bloqueantes. Adicionalmente a essas funções, algumas rotinas de ambiente, relativas a criação e finalização de processos, têm importância fundamental na implementação dos mecanismos necessários ao tratamento de mensagens em uma aplicação MPI básica. Essas funções estão dispostas no Apêndice A nas tabelas A.1, A.2 e A.3.

2.4.3 MPJ-Express

Nas implementações desenvolvidas nesse trabalho, foi utilizada uma ferramenta *open source* chamada MPJ-Express, que consiste em um conjunto de bibliotecas em Java que implementam as funcionalidades do MPI. Embora o MPI seja voltado para as linguagens C, C++ e Fortran, os desenvolvedores de programas paralelos podem aproveitar da alta portabilidade da linguagem Java, que permite, por meio da sua máquina virtual, executar aplicações em qualquer sistema operacional compatível. Isso amplia, sobremaneira, a capacidade de explorar o paralelismo de ambientes distribuídos, *grids*, *clusters* e *clouds*, além de dispor de soluções de aplicativos com um alto grau de abstração proporcionado pelas características da linguagem Java.

Segundo Shafi e Manzoor (2009), o MPJ-Express é uma implementação da API (*Application Programming Interface*) do mpiJava 1.2 (BAKER et al., 1999), e possui, como vantagem, maior praticidade na sua instalação e configuração em relação a outras implementações de MPI nativas de sistemas operacionais (SOs) Linux (AZEVEDO, 2012). Além disso, como consequência da alta portabilidade e independência de plataforma da linguagem Java, é possível configurar a ferramenta

para executar em aplicações com sistemas operacionais distintos, tanto em plataforma Windows (MPJ-EXPRESS, 2014b) como em plataforma Linux (MPJ-EXPRESS, 2014).

2.5 Considerações Finais

Este capítulo apresentou uma revisão bibliográfica a respeito dos temas importantes, e que servem de base para uma melhor compreensão do método, da implementação e dos experimentos, apresentados, necessariamente, nessa ordem nos capítulos seguintes. Neste capítulo foi realizado uma revisão extensa e detalhada sobre conceitos gerais de simulação, sobre os protocolos de sincronização otimistas *Time Warp* e *Rollback* Solidário e sobre a biblioteca de passagem de mensagens MPI. O capítulo seguinte apresentará o método para a análise de troca de protocolos desenvolvido neste trabalho e que é o cerne do mesmo.

3 Método para Análise de Troca de Protocolos Otimistas em um Ambiente de Simulação Distribuída

O método apresentado neste capítulo propõe uma solução genérica para a troca dinâmica entre os dois protocolos de sincronização otimistas, apresentados no capítulo anterior, no âmbito de um ambiente de simulação distribuída. Como visto no capítulo anterior, o protocolo *Time Warp*, desde sua elaboração, apresentou até hoje uma série de variações, no que diz respeito à forma como lida com o cancelamento de antimensagens, como realiza a coleta de lixo (*garbage colletion*), como é feito o salvamento e recuperação de estados, como é realizado o cálculo do GVT, como é realizado a operação de *rollback* (recuperação de erros de causa e efeito), como limita o otimismo do protocolo, entre outros fatores. Os trabalhos de Lobato (2001) e Lobato, Ulson e Santana (2004) propuseram uma taxonomia que agrupa essas diferentes variações do protocolo otimista *Time Warp*.

Entretanto, todas as variações são baseadas exclusivamente sobre o mesmo protocolo proposto por Jefferson (1985), e embora existam variações quanto à forma de lidar com problemas específicos do protocolo, a estrutura sobre a qual remonta o seu funcionamento é basicamente a mesma: relógio lógico para cada processo, estrutura de antimensagens, *rollback* local para cada processo podendo ou não um *rollback* de um determinado processo P_i influir em outro processo P_j, tal que $i \neq j$, $\forall i, j > 0$.

O protocolo *Rollback* Solidário, proposto por Moreira (2005), rompe com esses paradigmas e se apresenta como um outro tipo de protocolo otimista com caracte-

rísticas próprias. Como foi visto no capítulo anterior, o relógio lógico nos processos do *Time Warp* é substituído pela noção de relógio vetorial (FIDGE, 1991; MATTERN, 1989; SCHMUCK, 1988). A estrutura de antimensagens não existe mais, sendo substituída por um relógio matricial de ordem n, no qual n representa o número de processos da simulação, no processo observador, sendo este processo uma parte ativa durante o processo de *rollback* do protocolo *Rollback* Solidário, ao contrário da noção de *rollback* em cascata do *Time Warp*.

Considerando essas distinções, é certo afirmar que ambos os protocolos, *Time Warp* e o *Rollback* Solidário, são abordagens bastante distintas para protocolos otimistas. Entretanto, se os protocolos em questão são distintos entre si, como estabelecer critérios de comparação entre esses dois protocolos a fim de se elaborar um método que possa compará-los durante um espaço de tempo pré-definido? Como comparar um procedimento de *rollback* do *Time Warp* com um procedimento de *rollback* do *Rollback* Solidário, sendo que eles ocorrem de forma distinta? Como comparar o número de mensagens entre esses processos distintos?

Azevedo (2012) analisou a eficiência desses protocolos comparando o número de *rollbacks* realizados e a razão entre eventos perdidos e eventos salvos durante a execução da simulação até um determinado valor fixo do LVT para cada processo. Entretanto, devido à diferença entre os protocolos e suas respectivas particularidades surge a necessidade de um método melhor que estabeleça critérios mais apurados de comparação entre esses dois protocolos. Para este propósito, é necessário lançar mão de algumas definições formais que equiparem os protocolos, de modo que esse arcabouço teórico possibilite a formação de uma base comum de conceitos que possam sustentar essa comparação.

3.1 Definições

A possibilidade de comparação entre os protocolos distintos em questão depende em primeiro lugar do estabelecimento de parâmetros em comum entre si, pelos quais ambos possam ser avaliados e, a partir deles, possam determinar métricas que se apliquem igualmente tanto a um quanto ao outro protocolo. Como visto

anteriormente, ambos os protocolos de sincronização são otimistas, o que significa que ambos não evitam a ocorrência de erros de causa e efeito, ambos realizam a operação conhecida como *rollback* na reparação destes erros e, na execução desta operação, mensagens de comunicação entre os processos da simulação precisam ser enviadas, devido ao caráter distribuído da computação, para que a sincronização seja efetuada. Porém, essas semelhanças em si não são suficientes para que métricas possam ser aplicadas de forma absolutamente confiável na comparação entre os dois protocolos, pois não leva em consideração as particularidades das operações de *rollback* de cada protocolo e nem a forma como a comunicação interprocesso ocorre.

Tendo em vista o problema das características particulares de cada protocolo, e visando definir parâmetros que abarquem uma definição aplicável a ambos os protocolos, serão tomadas as seguintes definições:

- *Rollback* Local
  - *Rollback* Local Inicial
  - *Rollback* Local Induzido
- *Rollback* Global
- Mensagem de *Rollback*
- Intervalo de Simulação

3.1.1 Rollback Local

Os processos envolvidos na simulação, tanto do *Time Warp* quanto do *Rollback* Solidário, realizam a sua operação de *rollback* individual, referente a apenas um dado processo P_i ($\forall i > 0$) pertencente à simulação, de modo que, o processo em questão desfaz a computação errônea, retornando a computação para um estado seguro anterior a falha, e recupera estados anteriormente salvos. Desta forma:

Definição 1 *Define-se rollback local como o rollback referente a apenas um processo P_i. Adotar-se-á a notação r/P_i para um dado rollback que ocorra no processo P_i, $\forall i > 0$, tal que P_i pertença à simulação.*

Um *rollback* local pode ocorrer, em ambos os protocolos, devido a duas razões principais: por causa de uma mensagem *straggler* que gere um erro de causa e efeito na computação distribuída ou induzido por conta das próprias particularidades do procedimento de *rollback* do protocolo em questão. Com base neste fato, seguem-se as definições de *rollback* inicial e *rollback* induzido, analogamente, e respectivamente, às definições de *rollback* primário e *rollback* secundário para o *Time Warp*.

3.1.1.1 Rollback Local Inicial

Definição 2 *Define-se rollback local inicial, todo rollback local r/P_i que se inicie devido a uma mensagem que gere um erro de causa e efeito (mensagem straggler) no mesmo processo, de modo que, cada uma dessas mensagens marcam a ocorrência de um, e apenas um, desses rollbacks.*

3.1.1.2 Rollback Local Induzido

De maneira análoga à definição anterior, pode-se definir a noção de *rollback* induzido, como:

Definição 3 *Define-se rollback local induzido, todo rollback local r/P_j que se inicie induzido, direta ou indiretamente, por um rollback inicial r/P_i ($\forall i, j \mid i > 0, j > 0$ e $i \neq j$) de acordo com a particularidade de cada protocolo de sincronização.*

3.1.2 Rollback Global

Um procedimento de *rollback* local em um processo de uma computação distribuída pode acarretar a ocorrência de outros *rollbacks* locais em outros processos. No caso do *Time Warp*, um processo pode acarretar a ocorrência de *rollbacks*

em outros processos que possuam vínculo de dependência causal na computação distribuída desfeita pelo primeiro *rollback* local. No *Rollback* Solidário todos os processos são, até certo ponto, afetados quando um processo necessita realizar o seu *rollback* local. Desta forma, segue-se a próxima definição.

Definição 4 *Define-se rollback global R como o conjunto de dois ou mais rollbacks locais, no qual exista relação de causalidade entre ao menos um deles com os demais, de modo que, o rollback local inicial é o responsável pelo desencadeamento, direto ou indireto, de um ou mais rollbacks locais induzidos. Em outras palavras, um rollback global é um conjunto R que contém dois ou mais rollbacks locais referentes a processos distintos de modo que* $\exists r/P_i \in R$ *que desencadeie, direta ou indiretamente, um ou mais* r/P_j*, tal que* $\bigcup r/P_j \in R$*, para* $i \neq j$*.*

A figura 3.1 ilustra duas situações de *rollback* nas quais é possível identificar os conceitos de *rollback* local, global, inicial e induzido. Na figura é apresentado dois diagramas espaço-tempo, os quais o da esquerda representa um recorte de uma simulação usando o *Time Warp* e o da direita usando o *Rollback* Solidário. Os pontos em destaque nas linhas horizontais representam eventos, ao passo que as linhas transversais representam mensagens de um processo para outro.

3.1.3 Mensagem de Rollback

Durante um *rollback* global, mensagens de sincronização precisam ser enviadas para que esta operação se efetue. A essas mensagens dar-se-á o nome de mensagens de *rollback*. No *Time Warp*, as mensagens que se sucedem após o início de um *rollback* global são as antimensagens, ao passo que no *Rollback* Solidário as mensagens que se sucedem após o início de um *rollback* global são as mensagens de sincronização com o processo observador para o resgate da linha de recuperação ao qual a simulação deve retornar. Desta forma, segue-se a próxima definição.

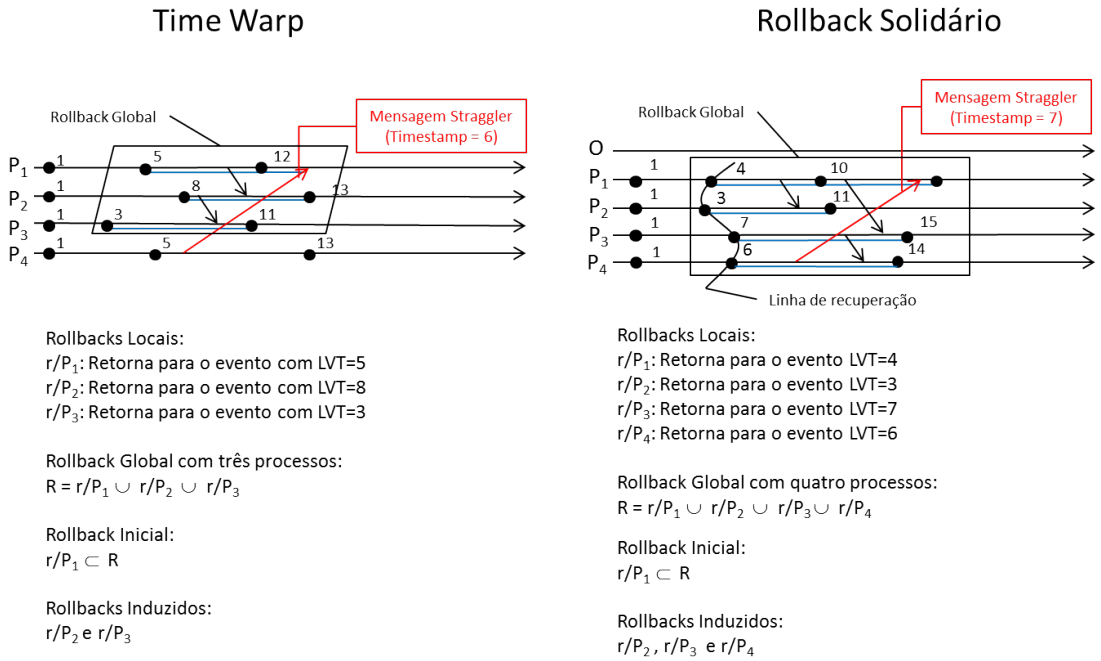

Figura 3.1: Exemplo de *rollbacks* locais, globais, iniciais e induzidos

Definição 5 *Uma mensagem de rollback é tal que:*

$$Mensagem\ de\ Rollback = \begin{cases} antimensagem,\ Time\ Warp \\ mensagem\ de\ comunicação\ processo/observador, \\ Rollback\ Solidário \end{cases}$$

3.1.4 Intervalo de Simulação

Uma simulação pode ser executada durante um tempo finito ou *ad-infinitum*. No caso da simulação estudada neste trabalho, esta ocorre em tempo finito, ou seja, ela terá um começo, meio e fim, e gerará valores de saída que expressem características a respeito do modelo executado (no caso deste trabalho, a respeito do protocolo de sincronização utilizado). Com isso, é possível definir intuitivamente intervalo de simulação como o espaço de tempo em que uma simulação, ou parte dela, ocorre. Quando o espaço de tempo de um intervalo não corresponde ao tempo total da simulação pode-se designá-lo como subintervalo de simulação.

Entretanto, quando se fala em "espaço de tempo em que uma simulação, ou parte dela, ocorre", para uma simulação distribuída, é preciso ter em conta que inexiste um relógio físico global que sincronize os processos que a constituem e, deste modo, cada processo, cuja execução pode se dar em processadores e máquinas diferentes entre si, terá seu próprio relógio local. Porém, quando se trata de apenas um processo, esta definição pode ser aplicada sem maiores problemas, pois o tempo de execução de um processo ocorre de forma linear. Baseado nisto, pode-se estabelecer uma definição formal para intervalo de simulação que estenda o conceito de forma que abarque os vários processos que fazem parte da simulação.

Definição 6 *Chama-se intervalo de simulação, ou subintervalo de simulação, o conjunto dos intervalos de todos os processos* $P_1, P_2, P_3, \ldots, P_n$ *pertencentes a simulação, excetuado o processo observador, decorrido um determinado tempo* t_i *de modo que* $1 \leq i \leq n$ *e* n *representa o número de processos da simulação. Quando um evento é computado dentro de um determinado intervalo de tempo de um processo específico, seja por meio de um rollback, ou por meio da chegada de uma mensagem, ou gerado pelo andamento da simulação, pode-se dizer que esse evento ocorre dentro do intervalo de simulação que contém esse intervalo de tempo.*

A figura 3.2 ilustra graficamente o que corresponderia a um subintervalo de simulação como sendo o conjunto dos subintervalos de tempos em cada um dos processos que compõem a simulação. Na ilustração, o intervalo K, precedido pelo intervalo $k-1$ e sucedido pelo intervalo $k+1$, é composto pela união dos intervalos dos processos $P_1, P_2, P_3, \ldots, P_n$ que compõem a simulação distribuída que está sendo executada nos tempos $t_1, t_2, t_3, \ldots, t_n$, respectivamente. Os tempos $t_1, t_2, t_3, \ldots, t_n$ de cada processo podem ser iguais ou diferentes entre si, entretanto, para a avaliação metodológica proposta neste trabalho, optou-se por fixá-los a priori com o mesmo tamanho.

Na implementação utilizada neste trabalho, cada processo anota informações internas relevantes ao desempenho da simulação que serão usadas nas métricas de comparação dos subintervalos de simulação. Tais informações podem ser elencadas como:

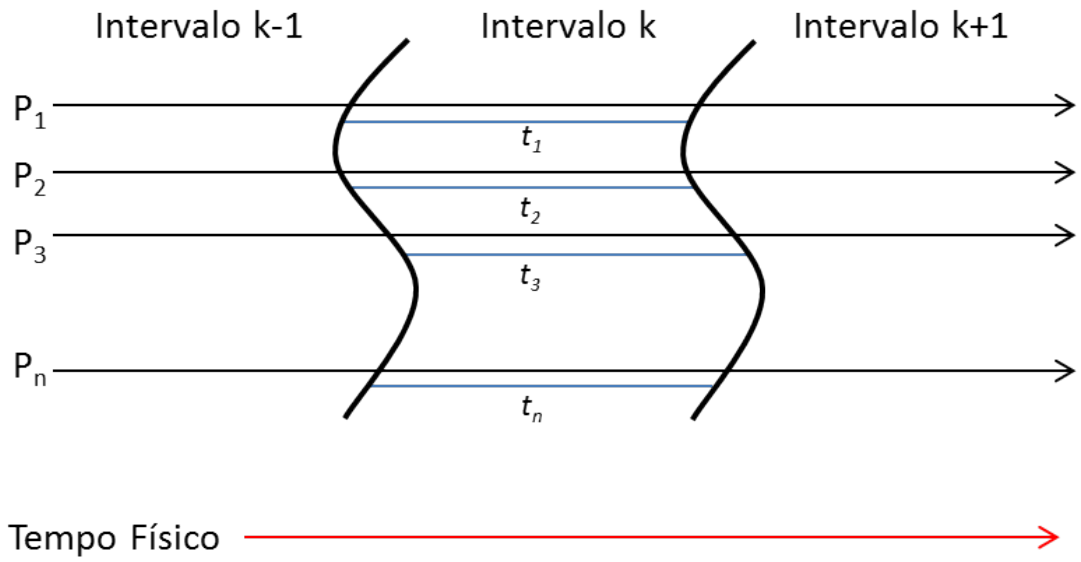

Figura 3.2: Intervalo de simulação k

- Número de eventos processados: o número de eventos que foram corretamente processados e não foram desfeitos por nenhuma operação de *rollback* ao final de cada subintervalo em um determinado processo;
- Número de eventos perdidos: o número de eventos que foram desfeitos por operações de *rollback* ao final de cada subintervalo em um determinado processo;
- Número total de eventos: a soma dos eventos salvos e os eventos perdidos;
- Número de *rollbacks* locais: o número de *rollbacks* locais ocorridos no referido processo.
- Número de mensagens *stragglers*: o número de mensagens *stragglers* recebidas pelo processo.
- Número de mensagens de *rollback*: o número de mensagens de *rollback* recebidas pelo processo.

Um subintervalo de simulação de tempo fixo pode apresentar o problema de acabar não contando corretamente alguns *rollbacks* locais, eventos perdidos e al-

gumas mensagens de *rollback* dentro do intervalo no qual o *rollback* global, ou local, que os originou tenha sido computado. Essa coerência na contagem precisa ser assegurada, pois um *rollback* deve ser usado como um dado que reflita o desempenho da simulação no intervalo em que ocorre, portanto não faz sentido que os *rollbacks* locais (no caso do *rollback* originário ser global), os eventos perdidos e as mensagens de *rollback* decorrentes de um *rollback* que ocorreu em um dado intervalo, sejam computados em um intervalo de simulação posterior. Esses dados devem ser registrados como partes do mesmo intervalo da mensagem *straggler* que desencadeou o *rollback* inicial correspondente a essas operações secundárias.

A figura 3.3 retrata uma mensagem *straggler* que chega do processo P_3 para o processo P_1 e, com isso, ocasiona um *rollback* neste processo que resultará por desfazer uma computação errônea até certo ponto e essa operação levará um determinado tempo. Deve-se notar que embora um *rollback* restaure uma computação para um estado anterior ao *timestamp* da mensagem *straggler*, e no diagrama espaço-tempo essa computação restaurada esteja representada à esquerda da mensagem no diagrama, o tempo físico jamais retrocede, e por isso, o período de tempo para que uma computação errônea seja desfeita é representada à direita da chegada da mensagem, ultrapassando assim o limite entre o intervalo k e o intervalo $k+1$. Essa situação é particularmente problemática, pois a parte da computação desfeita no intervalo $k+1$ se deve a um *rollback* que se originou no intervalo k e, portanto, o registro desses dados deve se dar no intervalo aonde se iniciou.

Para resolver esse problema, basta que a cada vez que uma mensagem *straggler* chegar a um processo e desencadear um *rollback*, o intervalo no qual a mensagem foi recebida seja anotado. Todos os dados da computação que retroceder por conta desse *rollback*, tanto no próprio processo que recebeu a mensagem, quanto nos processos afetados pelos respectivos *rollbacks* induzidos do *rollback* global desencadeado, serão registrados como ocorridos dentro do intervalo no qual o *rollback* inicial se iniciou.

Na seção seguinte são definidas as métricas utilizadas no método de comparação de intervalos apresentado neste trabalho na seção 3.3. Estas métricas são calculadas a partir das informações de desempenho anotadas por cada processo durante o decorrer da simulação para cada subintervalo.

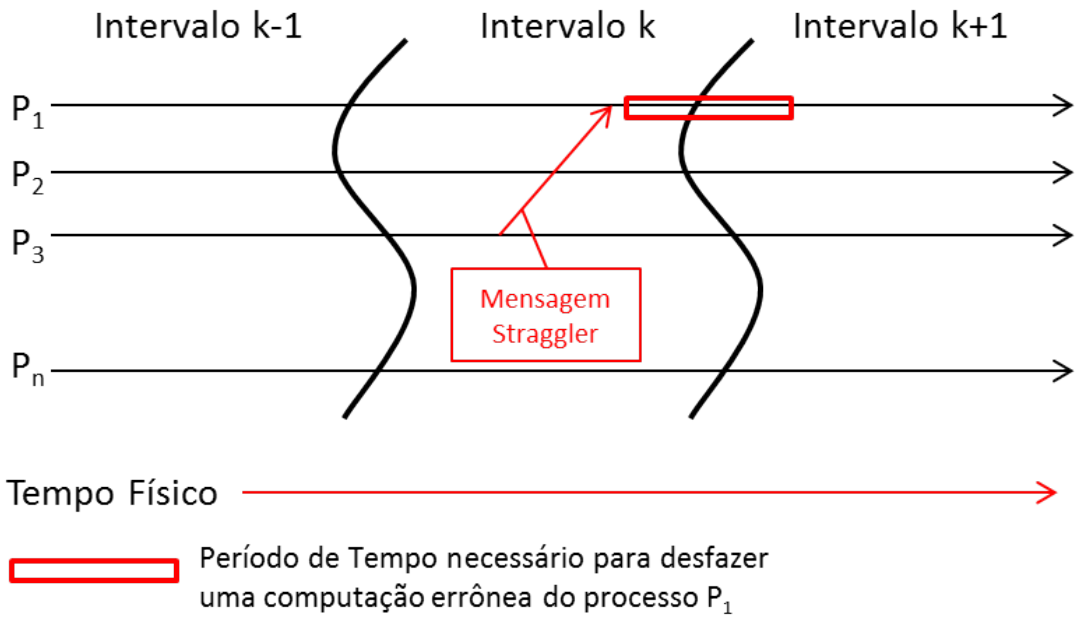

Figura 3.3: Interferência entre intervalos

3.2 Métricas

Devido às características totalmente distintas dos protocolos, *Time Warp* e *Rollback* Solidário, há uma dificuldade inerente para se traçar um comparativo direto de desempenho entre eles baseado apenas na quantidade de *rollbacks* efetuados durante intervalos de simulação com tempos iguais. Esse problema se deve ao fato de que a operação de *rollback* em ambos os protocolos é diferente, e não faz sentido comparar apenas a quantidade de operações de recuperação de causa e efeito sem levar em conta as suas particularidades.

As definições criadas anteriormente se aplicam de forma equivalente aos dois protocolos, de modo que é possível agora aplicar algumas métricas definidas no trabalho de Lobato (2001), redefinindo-as com base nas definições anteriores, no método para comparar os resultados de dois intervalos de simulação em um mesmo intervalo de tempo relativo, de modo que possam se estabelecer critérios seguros para se avaliar qual protocolo teve desempenho melhor ou pior em um dado intervalo de tempo. As métricas adotadas totalizam em cinco, sendo a última usada como critério de desempate e são detalhadas a seguir.

- Tamanho médio do *Rollback*: o tamanho médio do *rollback*, representado na equação 3.1, é expresso na quantidade média de eventos desfeitos a cada *rollback*. Como cada *rollback* global é um conjunto de um ou mais *rollbacks* locais, pode-se dizer que o tamanho médio do *rollback* pode ser calculado como o número total de eventos desfeitos pelo número total de *rollbacks* locais.

$$TMR = \frac{\text{nº eventos perdidos}}{\text{nº rollbacks locais}} \tag{3.1}$$

- Frequência de *Rollbacks*: a frequência de *rollbacks*, representada na equação 3.2, expressa o quanto o andamento da simulação, e as características do modelo, afetam a ocorrência de *rollbacks*. Neste caso, considera-se a frequência de *rollbacks* como o número de *rollbacks* iniciais, pois estes são causados pelas circunstâncias da simulação, dividido pelo número total de eventos (processados e desfeitos). Não obstante, uma mensagem *straggler* que chegue a um determinado processo causará, necessariamente, um *rollback*. Deste modo, pode-se contar o número de *rollbacks* iniciais pelo número total de mensagens *stragglers* ocorridas durante a simulação para todos os processos.

$$FR = \frac{\text{nº de mensagens } \textit{stragglers}}{\text{nº total de eventos}} \tag{3.2}$$

- *Rollbacks* por Eventos: esta métrica, representada na equação 3.3, expressa qual a relação entre a quantidade de *rollbacks* ocorridos e a quantidade de eventos processados. Neste caso, a métrica *rollbacks* por eventos é o número de *rollbacks* locais dividido pelo número total de eventos processados.

$$RE = \frac{\text{nº de } \textit{rollbacks} \text{ locais}}{\text{nº de eventos processados}} \tag{3.3}$$

- Eficiência: a eficiência, representada na equação 3.4, é o número total de eventos processados pelo número total de eventos.

$$\text{Eficiência} = \frac{\text{nº de eventos processados}}{\text{nº total de eventos}} \tag{3.4}$$

- Mensagens de *Rollback*: trata-se de comparar o número das mensagens de *rollback* em dois subintervalos.

As métricas descritas podem ser aplicadas a um processo ou para todos os processos envolvidos na simulação, de modo que para aplicá-las ao intervalo de simulação seria necessário calcular o somatório dos dados de desempenho de cada um dos processos envolvidos.

3.3 Um método para Troca de Protocolos

Uma vez definidas as métricas que serão usadas para se comparar os subintervalos de simulação dos protocolos, cabe agora definir como a comparação entre esses subintervalos de diferentes protocolos se efetua, e porque o método apresentado neste trabalho leva a um resultado otimizado de execução em termos de desempenho. Estes passos, consecutivamente, somados a avaliação do custo da troca de protocolos entre intervalos de simulação constituem o cerne deste trabalho.

A primeira parte deste método consiste na avaliação e decisão sobre a mudança ou não de um protocolo para outro durante a simulação, que é feita por um algoritmo guloso que avalia dois subintervalos de simulação de um mesmo intervalo de tempo relativo e decide preliminarmente qual dos dois é mais apto a ser executado com base nas métricas definidas na seção anterior para se comparar esses intervalos. Um algoritmo guloso sempre faz a escolha que parece ser a melhor no momento, isto é, ele faz uma escolha ótima para as condições locais na esperança de que essa escolha leve a uma solução ótima para a situação global (CORMEN et al., 2012). Este método funciona basicamente como o processo de escolha do menor caminho para uma árvore binária com peso nas arestas, pois cada decisão local sobre a mudança ou não de protocolo no início de cada subintervalo gera uma possibilidade de execução híbrida como solução global, que teoricamente reflete a possibilidade de máximo desempenho entre os protocolos.

Uma vez que se obtenha uma possibilidade de execução híbrida teórica com base nas métricas propostas, caberia então validar se os custos reais da troca de protocolos ao final de cada possibilidade de troca definida na execução híbrida

compensariam, ou não, os ganhos em termos de desempenho obtidos com a troca. Com base no resultado dessa análise de custos, é possível verificar se o modelo de otimização teórico obtido na primeira etapa converge, e o quanto converge, na obtenção da computação de melhor desempenho na prática por meio da execução de diferentes modelos de simulação. Essa segunda parte do trabalho, a avaliação dinâmica do modelo de execução híbrida proposta na primeira parte, é definida em termos teóricos na seção 3.4 e explanada nos estudos de caso do capítulo 5.

O método proposto neste trabalho consiste em verificar em quais condições é vantajoso se executar a troca dinâmica de protocolos por meio do teste de um modelo teórico definido na primeira etapa, de modo a conferir se o custo das trocas de protocolo entre intervalos são menores que os ganhos obtidos com a troca de protocolos. Caso seja, a troca de protocolos seria efetuada.

3.3.1 Comparação de Intervalos

Inicialmente, para que os intervalos possam ser comparados, é necessário assegurar que os dados de desempenho registrados em cada subintervalo de simulação não sofram influência de nenhum outro anterior ou posterior a ele, pois cada um deles deve ser compreendido como uma unidade isolada, um recorte consistente dentro da simulação, no qual os dados gerados durante este subintervalo reflitam o estado da simulação naquele intervalo de tempo, independente se as computações realizadas nos demais intervalos foram executadas pelo mesmo protocolo do subintervalo, ou se houve troca de protocolos nos subintervalos de simulação adjacentes. A computação ocorrida nos intervalos adjacentes não deve influenciar o registro de dados de desempenho durante um subintervalo de simulação, de modo a causar dependência entre eles.

Para assegurar que *rollbacks* não interfiram em intervalos adjacentes, basta que se implemente o registro de dados de desempenho da forma como foi discutida na subseção 3.1.4. A cada mudança de intervalo, um algoritmo de cálculo do GVT recebe as mensagens transientes e assegura que todas elas sejam computadas, com todos os seus efeitos subsequentes, dentro do intervalo de simulação que se originaram. Esse algoritmo será mais bem detalhado na subseção 3.3.3.

Cada subintervalo contém em si uma possibilidade de execução de um protocolo distinto, no caso deste trabalho de dois protocolos, *Time Warp* e *Rollback* Solidário, embora nada impeça que o modelo deste método possa ser expandido para abarcar outras otimizações diferenciadas desses protocolos otimistas. Uma vez assegurada a independência entre os intervalos, é possível definir como essas comparações se dão e como o método funciona.

Uma mesma simulação, com intervalos de tempo previamente definidos, executada para os protocolos *Time Warp* e *Rollback* Solidário, gerará duas possibilidades de execução. Fazendo os intervalos de simulação de cada linha de execução com limites idênticos, pode-se estabelecer um comparativo entre os subintervalos equivalentes e escolher entre eles qual teve o melhor desempenho segundo as métricas definidas. A partir deste comparativo, é possível construir uma linha de execução híbrida, que envolva em sua composição tanto intervalos executados pelo protocolo *Time Warp* quanto intervalos executados pelo protocolo *Rollback* Solidário. A figura 3.4 ilustra um exemplo deste comparativo, no qual os intervalos TW_x, de uma linha de execução utilizando o protocolo *Time Warp*, são comparados com os intervalos de uma linha de execução utilizando o *Rollback* Solidário SR_x.

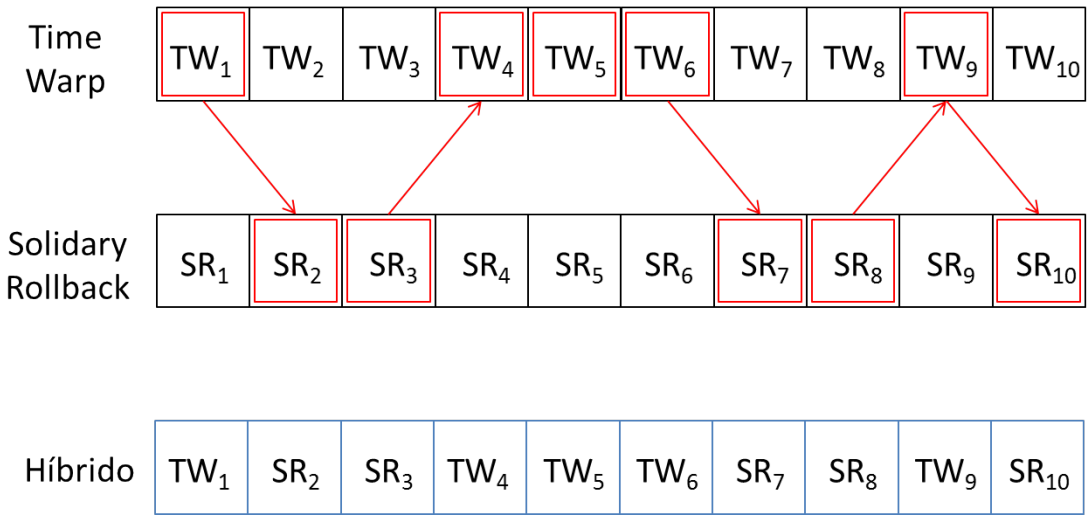

Figura 3.4: Comparativo entre duas linhas de execução diferentes

O algoritmo guloso analisa os intervalos da esquerda para a direita, comparando primeiramente TW_1 com SR_1, em seguida, TW_2 com SR_2 e assim sucessi-

vamente até chegar ao índice 10 (número máximo de intervalos). A solução local, após a comparação de cada subintervalo equivalente, é agregada na construção da solução global que resulta na linha de execução híbrida ilustrada na figura. Esse comparativo é feito por meio das cinco métricas definidas na seção 3.2. Estas são calculadas para cada intervalo a partir dos dados de desempenho registrados, e comparadas para cada intervalo equivalente. O estabelecimento de um número ímpar de métricas é importante, pois isso evita empates durante a comparação.

Mudanças de protocolos envolvem custos de trocas, estes denotados pelas setas da linha de execução de um protocolo para o outro na figura 3.4. Considerá-los no cálculo efetivo para a troca de protocolos é essencial para se verificar se a linha de execução híbrida, encontrada neste primeiro passo, realmente apresenta a melhor solução para a troca de protocolos dinâmica, ou se o custo da troca é maior que a diferença de ganho baseada nas métricas, sendo mais vantajoso em termos de desempenho continuar executando o mesmo protocolo.

Além do cálculo das métricas, outro dado é obtido a cada intervalo: a média do tempo total da computação desperdiçada TD_k. Trata-se da somatória da quantidade de tempo td_i que cada processo P_i utiliza fazendo *rollback*, tanto locais quanto globais. Este dado, embora obtido de modo independente do cálculo de métricas, é afetado de modo direto por elas e isto será provado na seção seguinte. O cálculo deste valor, expresso na fórmula 3.5, busca pré-validar o real desempenho da linha de execução híbrida gerada comparando os diferentes valores TD_k entre *Time Warp* e *Rollback* Solidário e verificando se estes resultados e o resultado da linha híbrida inicial convergem para a mesma conclusão.

$$TD_k = \sum_{i=1}^{n} \frac{td_i}{n} \tag{3.5}$$

A validação mencionada no parágrafo anterior é expressa na figura 3.5.

A seção 3.4 mostra como calcular custos de troca e como comparar esses valores de maneira alheia às métricas propostas, de modo a validar se essas de fato são um indicativo válido se a simulação está convergindo.

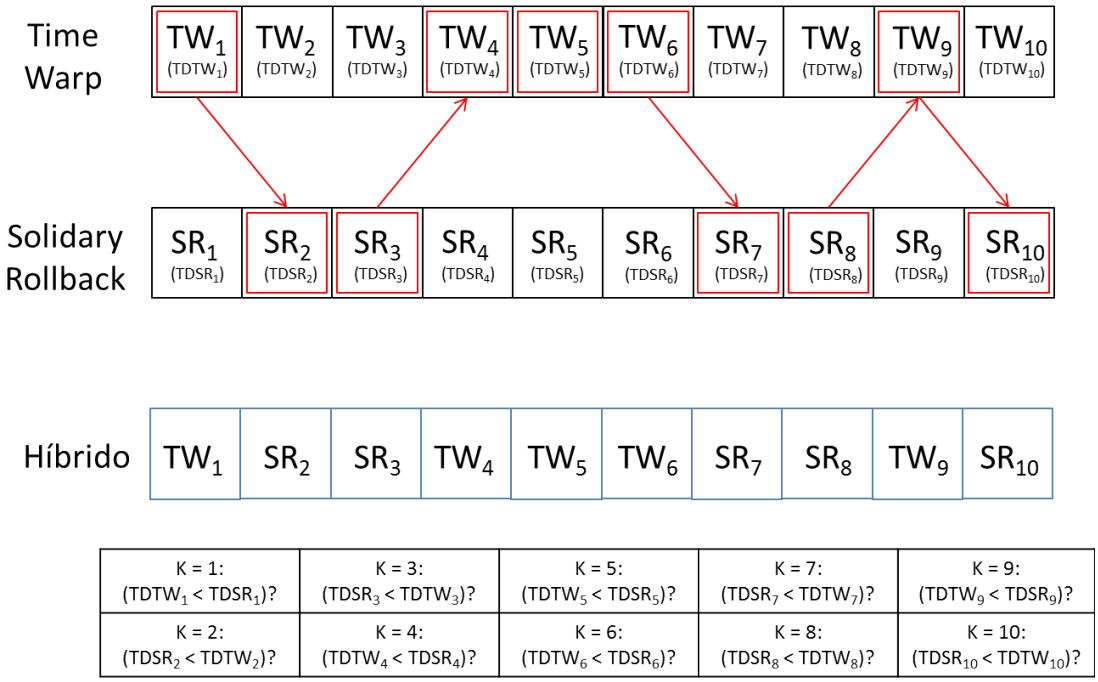

Figura 3.5: Pré validação da linha híbrida com base no TD_k

3.3.2 Convergência de Métricas

As métricas são válidas se refletem, de alguma forma, o valor de cada td_i para cada processo P_i, refletindo consecutivamente o valor de TD_k. Cada uma delas mede, de forma direta ou inversamente proporcional, o cálculo de TD_k de modo bastante evidente.

O tamanho médio do *rollback* (equação 3.1) é calculado a partir de quantos eventos em média são desfeitos a cada *rollback*. Quanto maior o TMR, maior a quantidade de eventos desfeitos em relação à quantidade de *rollbacks*. Um *rollback* para existir precisa ter no mínimo um evento perdido, caso contrário não há computação desfeita, portanto o caso quase ótimo seria o $TMR = 1$, que significaria necessariamente um evento desfeito a cada *rollback*. O caso ótimo seria o $TMR = 0$, porém essa situação particular é apontada nos relatórios quando não há existência de *rollbacks*, nem de eventos perdidos no sistema, e seria o caso de um processo que não receba mensagens de nenhum outro processo, ou no caso de processos adequadamente sincronizados, e, portanto, não desencadeie nenhum

procedimento de computação desfeita. Posto isto, quanto maior o TMR, maior a quantidade de eventos desfeitos para um número fixado de *rollbacks* locais, e quanto mais eventos desfeitos, maior a quantidade de tempo necessária para desfazer a computação errônea em questão. Portanto, o crescimento do TMR é diretamente proporcional ao crescimento do TD_k.

A frequência de *rollbacks* (equação 3.2) é calculada a partir da quantidade de mensagens *stragglers* (expressam a quantidade de *rollbacks* iniciais) recebidas para a quantidade total de eventos, processados e perdidos, o que a torna uma medida relativa. Entretanto, quanto maior a quantidade de mensagens *stragglers* maior a quantidade de *rollbacks*, e quanto maior a quantidade de *rollbacks* maior a quantidade de eventos perdidos. Em contrapartida, o número de eventos processados corretamente diminui a cada operação de *rollback*. Como o total de eventos é igual ao número de eventos processados mais a quantidade de eventos perdidos, a medida do divisor se mantém inalterada a cada novo *rollback* inicial causado por uma mensagem *straggler*. Portanto, quanto maior a frequência de *rollbacks* iniciais, maior o tempo que será despendido com computações errôneas. Portanto, visto que todos os intervalos tem o mesmo tamanho limite de tempo de duração, o crescimento da FR é diretamente proporcional ao crescimento do TD_k.

A métrica *rollbacks* por eventos (equação 3.3) é calculada pelo número de *rollbacks* locais dividido pelo número total de eventos processados. Quanto maior o número de *rollbacks* locais, menor o número de eventos processados. O crescimento da RE é diretamente proporcional ao crescimento do TD_k.

A eficiência (equação 3.4) é o número total de eventos processados, dividido pelo número total de eventos. Quanto maior o número de *rollbacks*, maior o número de eventos perdidos, menor o número de eventos processados, ao passo que o número total de eventos se mantém constante. Similarmente ao que ocorre com a métrica frequência de *rollbacks*, quanto menor o número de eventos processados, menor a eficiência. O crescimento da eficiência é inversamente proporcional ao crescimento do TD_k.

As mensagens de *rollback* equivalem no protocolo *Time Warp* as antimensagens necessárias no *rollback* em cascata, que é a técnica que descreve seu processo

de *rollback* global, ao passo que correspondem às mensagens de sincronização dos processos da simulação com seu observador no *Rollback* Solidário. Um aumento no número de antimensagens no *Time Warp* significa um aumento no número de *rollbacks* induzidos, ao passo que o número de mensagens de sincronização no *Rollback* Solidário significa mais processos dentro de um estado de *rollback*. Portanto, o crescimento do número de mensagens de *rollback* é diretamente proporcional ao crescimento do TD_k.

Embora todas as métricas influenciem diretamente no TD_k é necessário analisa-las em conjunto para que não se forme uma impressão errada a respeito do desempenho da simulação, resultante de uma análise parcial e isolada. Por exemplo, um número pequeno de mensagens de *rollback* durante um intervalo pode ser devido a uma latência na rede muito grande que esteja atrasando a transmissão de mensagens. Neste caso, embora isso possa reduzir o número de *rollbacks*, isso não simboliza necessariamente uma melhora de desempenho, pois um atraso muito grande na transmissão de mensagens de *rollback* ou de mensagens *stragglers* pode acarretar no aumento de eventos desfeitos por *rollback*, aumentando assim o tamanho médio do *rollback* e consecutivamente aumentando o TD_k, ao invés de diminuí-lo.

3.3.3 Mecanismo de Cálculo de GVT

Para se evitar que um intervalo interfira nos seus intervalos adjacentes, é implementado um mecanismo de *checkpoint* global que trate todas as mensagens transientes antes de iniciar o registro de dados de desempenho para o intervalo seguinte. Este mecanismo de *checkpoint* global ao final de cada subintervalo tem três finalidades: (1) calcular o GVT da simulação ao final de cada intervalo; (2) processar todas as mensagens transientes, de modo que nenhuma mensagem *straggler* possa ultrapassar a fronteira do intervalo; (3) após o processamento de todas as mensagens transientes, definir uma linha de recuperação com LVTs atuais de cada processo, de modo que cada processo que constitui a simulação possa reiniciar a sua execução daquele ponto em diante. Cada protocolo implementa esse mecanismo de forma distinta, como será comentado no capítulo 4, entretanto a

noção básica do que ocorre pode ser ilustrada na figura 3.6.

As linhas ilustradas na figura 3.6 são a linha limite do intervalo k, a linha do GVT e a linha de LVTs atuais que inclui o valor do LVT para cada processo após o processamento das mensagens transientes. A linha limite do intervalo k contém os valores de LVT de cada processo após este ter executado o limite de tempo especificado para o respectivo intervalo. A linha de LVTs atuais contém os valores de LVT de cada processo após o processamento das mensagens transientes que ultrapassaram a linha limite de k, ao passo que a linha do GVT é uma linha que contém, para todos os processos, o valor do GVT que é calculado como o valor mínimo dos valores da linha de LVT atuais. A linha de LVTs atuais denota qual será o valor de LVT no qual a simulação retomará sua execução no intervalo $k+1$, ao passo que a linha do GVT denota o mínimo valor para o qual cada processo poderá retornar no caso de um *rollback*.

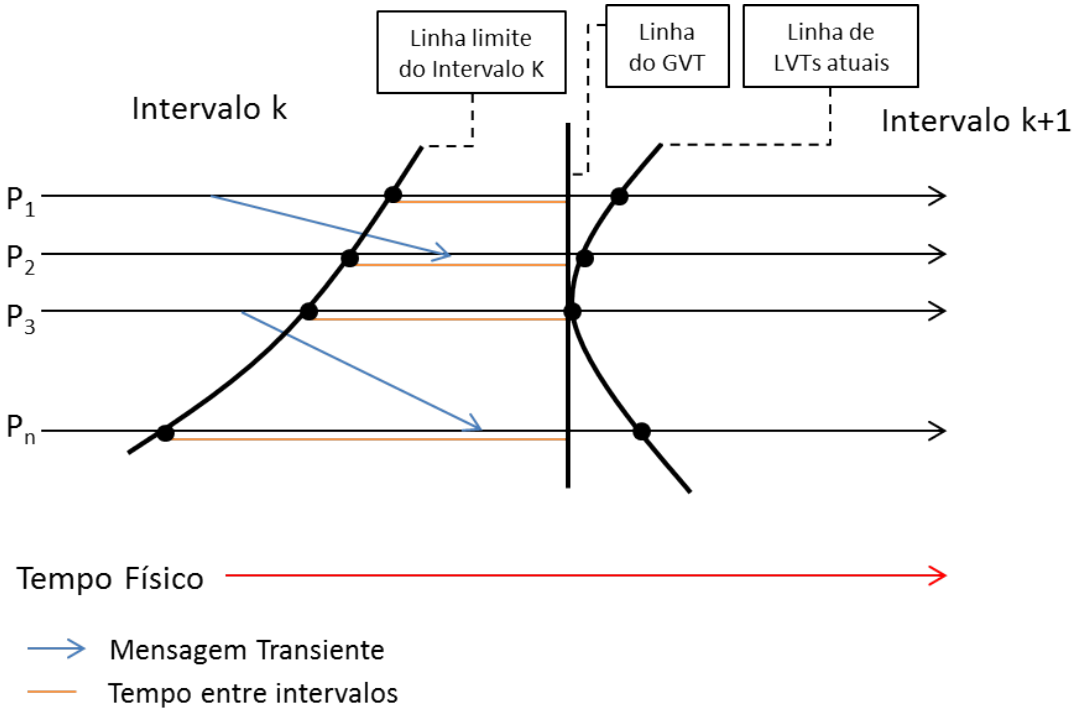

Figura 3.6: Mecanismos de *checkpoint* global entre intervalos

Independentemente se a simulação trocará ou não de protocolo de sincronização na troca de intervalo, este mecanismo é necessário a cada mudança de intervalo,

pois a finalidade do mesmo é manter a independência entre os subintervalos de simulação, assegurando que nenhuma mensagem transiente ultrapassará a fronteira do intervalo. É importante notar na figura 3.6 que a escala de tempo do diagrama é dada considerando o tempo físico da simulação e não o tempo lógico da mesma. Desta forma, a linha do GVT e a linha de LVTs atuais estão representadas necessariamente a direita da linha limite do intervalo k. Considerando apenas o tempo lógico da simulação, as linhas geradas pelo mecanismo poderiam estar à esquerda da linha limite do intervalo, uma vez que as mensagens transientes poderiam ser mensagens *stragglers* ou mensagens de *rollback* que poderão fazer a computação retornar para um LVT menor do que o LVT que possuía na linha limite inicial.

O tempo decorrido para que se produzam as linhas de GVT e a linha de LVTs atuais, não será levado em consideração no cálculo do custo de troca de protocolos, embora esse mecanismo seja necessário para que esta troca seja efetuada. Isto se deve ao fato de que essa operação será realizada de qualquer modo independente se o protocolo for alterado ou não, e este tempo servirá apenas para adicionar um acréscimo na contagem do próximo intervalo, para que não haja defasagem no tempo limite para cada intervalo.

3.4 Análise do Custo de Troca de Protocolos

Analisar o custo da troca de protocolos é importante em um segundo momento para que se possa verificar se a linha de execução híbrida, resultante da comparação de intervalos equivalentes por meio das métricas, converge em produzir um resultado de maior desempenho. O que é feito nesta etapa é comparar o custo em termos de tempo levado para se efetuar a troca de protocolos no sistema da simulação e o tempo resultante da diferença de desempenho entre intervalos equivalentes das duas linhas de execução entre os dois protocolos. Para que isso possa ser feito, é necessário definir como esses tempos serão calculados e como funciona a transição em conjunto entre intervalos para todos os processos da simulação.

O cálculo de custos virá após estas linhas estarem definidas. A figura 3.7 ilustra, por meio de quatro exemplos, possíveis transições de intervalos para um

mesmo processo P_x. Os dois primeiros casos ilustram um exemplo de continuidade usando o mesmo protocolo e os dois últimos casos ilustram um exemplo de mudança de protocolos entre intervalos.

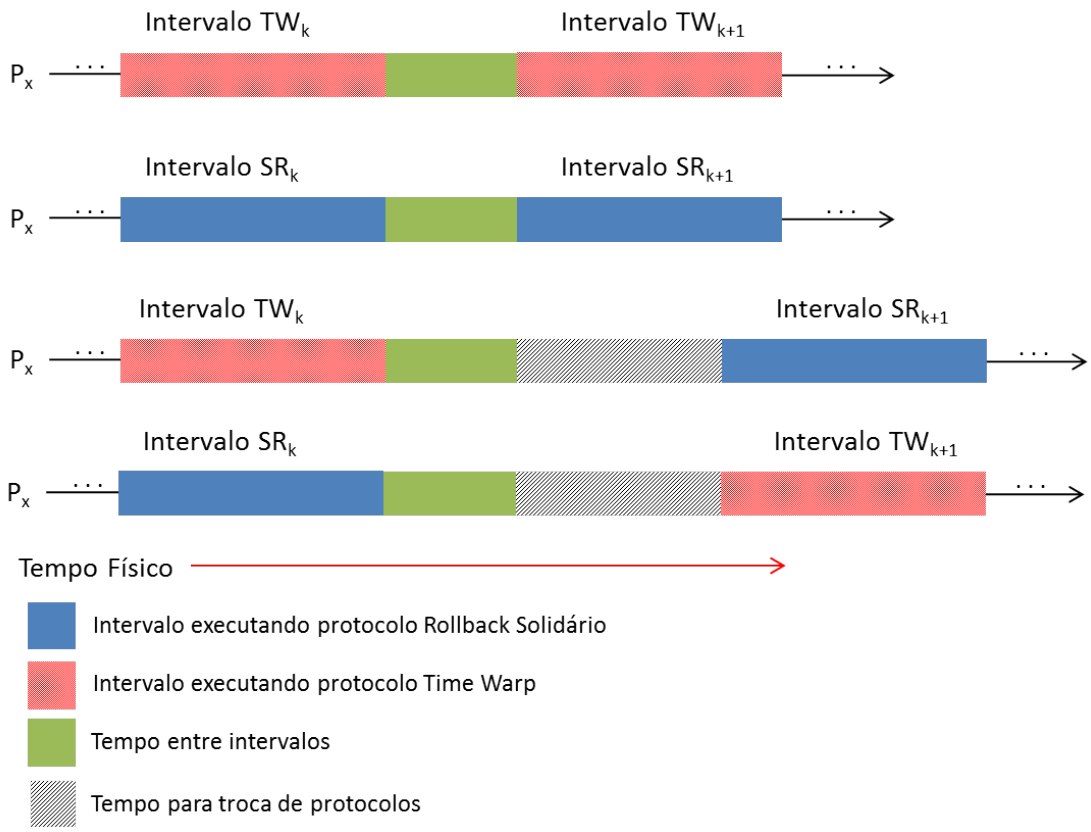

Figura 3.7: Possíveis situações de transição de intervalos

Uma vez estando essas linhas definidas, resta agora calcular o tempo necessário para a troca de protocolos durante a simulação. Isso envolverá somar o tempo necessário para que, tanto o processo observador quanto os processos pertencentes à simulação, sejam reiniciados usando o novo protocolo de sincronização. Cabe ressaltar que o *Time Warp* na sua concepção original (JEFFERSON, 1985) independe de processo observador, entretanto, tanto na implementação deste trabalho quanto na implementação do código herdado de Azevedo (2012), ele se faz necessário no momento de carregar o modelo de simulação e iniciar os demais processos, e na implementação do mecanismo de cálculo do GVT. Isso será detalhado melhor no capítulo seguinte.

O processo de troca de protocolo envolve recarregar, nos processos da simulação, as duas linhas de recuperação que demarcam o início do novo intervalo, a semente de números aleatórios, e a lista de eventos futuros, cujos eventos ainda não foram executados no intervalo anterior. O processo observador precisa ser recarregado também, pois, além de ser o responsável pela inicialização dos processos da simulação com os dados do modelo, este desempenha diferentes funcionalidades nos dois protocolos. Desta forma, pode-se calcular o custo total de troca entre os protocolos CT_k como sendo a média dos tempos de troca entre protocolos dos processos da simulação, conforme mostra a equação 3.6, na qual n representa o número de processos da simulação. O custo de troca é ilustrado no diagrama espaço-tempo na figura 3.8.

$$CT_k = \sum_{i=1}^{n} \frac{t_i}{n} \tag{3.6}$$

É importante observar que o tempo gasto pelo processo observador (P_0) para trocar de protocolos já está imbutido no tempo que cada processo da simulação levará para fazer a troca. Isso se deve ao fato de que deve haver sincronismo entre os processos ao final de cada intervalo para a troca de protocolos. Desta forma, o processo observador P_0 calculará o custo total CT_k com base nas informações de tempo para a troca de protocolos calculadas em cada processo, assim ao final de cada intervalo, cada processo P_i envia seu tempo de troca t_i juntamente com o seu tempo entre intervalos para o processo observador.

Quando não houver troca de protocolos, $t_i = 0$. Assim sendo, os custos de troca calculados do intervalo k, para o intervalo $k + 1$, serão repassados para o processo observador ao final do intervalo k. Esse método busca descobrir quando é vantajoso efetuar a troca de protocolos durante a simulação sem possuir de fato um mecanismo de comparação e troca dinâmica de protocolos. O que se visa aqui é testar um modelo de execução híbrido definido após algumas rodadas de simulação dos dois protocolos, e verificar se ele converge na prática considerando os custos da troca de protocolos. Este teste é realizado executando-se o modelo de simulação seguindo a linha de execução híbrida previamente estabelecida, efetuando-se a

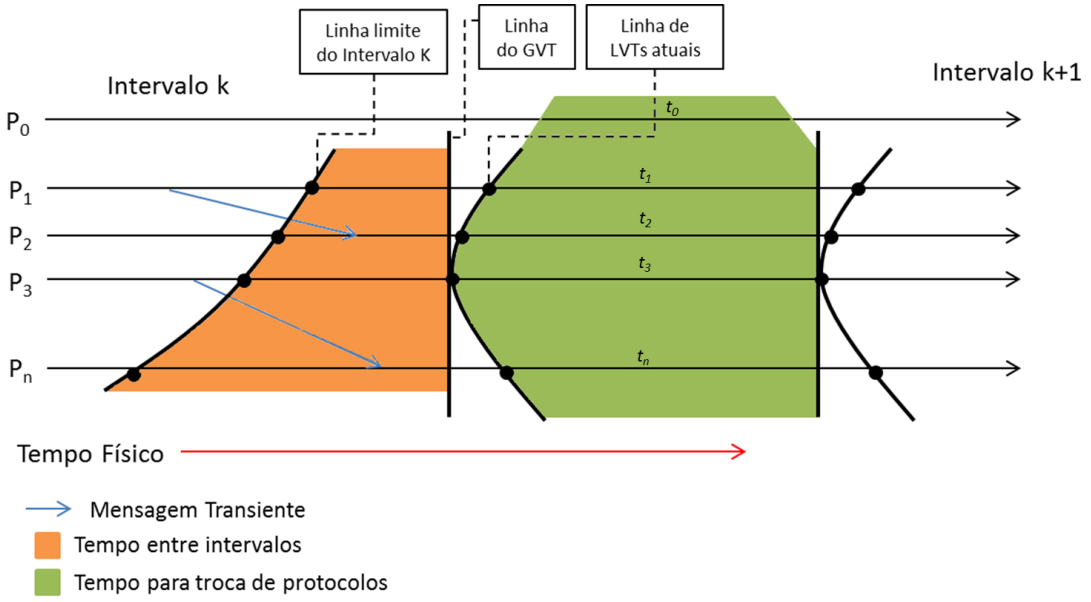

Figura 3.8: Tempos entre intervalos e para trocas de protocolos

troca de protocolos e o cálculo de custos quando for necessário.

Para se estabelecer o comparativo, consideram-se os CT_k e o TD_k da execução atual, e o TD_k do intervalo alternativo ao que se optou por efetuar a troca com base na linha de execução híbrida, que será denotado por TDA_k. A fórmula 3.7 expressa que se o custo de troca mais o tempo desperdiçado pela computação no intervalo k, denotado por $CT_k + TD_k$, for menor que o tempo desperdiçado pela alternativa de execução por outro protocolo TDA_k, então a troca é vantajosa. Caso contrário, a troca não é vantajosa.

$$\text{Mudança de Protocolo} = \begin{cases} \text{Sim}, & \text{se } TDA_k > CT_k + TD_k \\ \text{Não}, & \text{se } TDA_k \leq CT_k + TD_k \end{cases} \tag{3.7}$$

Um fator que não pode ser ignorado é o tempo gasto por cada processo na troca de intervalos. O mecanismo de cálculo do GVT pode levar tempos diferenciados para o cálculo do GVT e para a consequente transição de um intervalo para outro. Considerando que a simulação fica bloqueada durante este tempo, é necessário que este tempo seja incluso complementarmente na equação 3.7, adicionando-se ao

respectivo tempo desperdiçado, o tempo de transição TT_k, referente ao intervalo k em questão. Desta forma, pode-se reescrever a equação 3.7 da forma mostrada na equação 3.8.

$$\text{Mudança de Protocolo} = \begin{cases} \text{Sim}, & \text{se } (TDA_k + TTA_k) > CT_k + (TD_k + TT_K) \\ \text{Não}, & \text{se } (TDA_k + TTA_k) \leq CT_k + (TD_k + TT_K) \end{cases} \tag{3.8}$$

A possibilidade de se elaborar um método que verifique a vantagem de se executar uma troca de protocolos sem a existência de um mecanismo de comparação dinâmica de desempenho, se deve ao fato de que os custos envolvidos nessa comparação são precisamente irrelevantes no contexto da avaliação da troca, uma vez que essa comparação precisaria ser efetuada havendo ou não troca de protocolos.

3.5 Considerações Finais

O método apresentado neste capítulo verifica se é vantajoso que a simulação execute trocas de protocolos durante intervalos de tempo préviamente definidos, sem possuir necessariamente um mecanismo de comparação dinâmica. Desta forma, o método é composto de duas partes, nas quais ao final da execução das mesmas é possível verificar qual protocolo otimista, entre *Time Warp* e *Rollback* Solidário, é o mais indicado para sincronizar um determinado modelo de simulação durante um determinado intervalo de tempo específico.

A figura 3.9 traz uma representação esquemática, em forma de fluxograma, do funcionamento desse método.

A primeira parte consiste em subdividir o tempo de execução total da simulação, para cada protocolo, em intervalos de tempo previamente definidos, e compará-los de modo a obter uma possibilidade de execução híbrida, nas quais se intercalariam a execução de intervalos usando o protocolo *Time Warp* e outros usando o *Rollback* Solidário. Para definir qual o protocolo mais apto a rodar durante cada intervalo, são usadas métricas que avaliem o desempenho de ambos

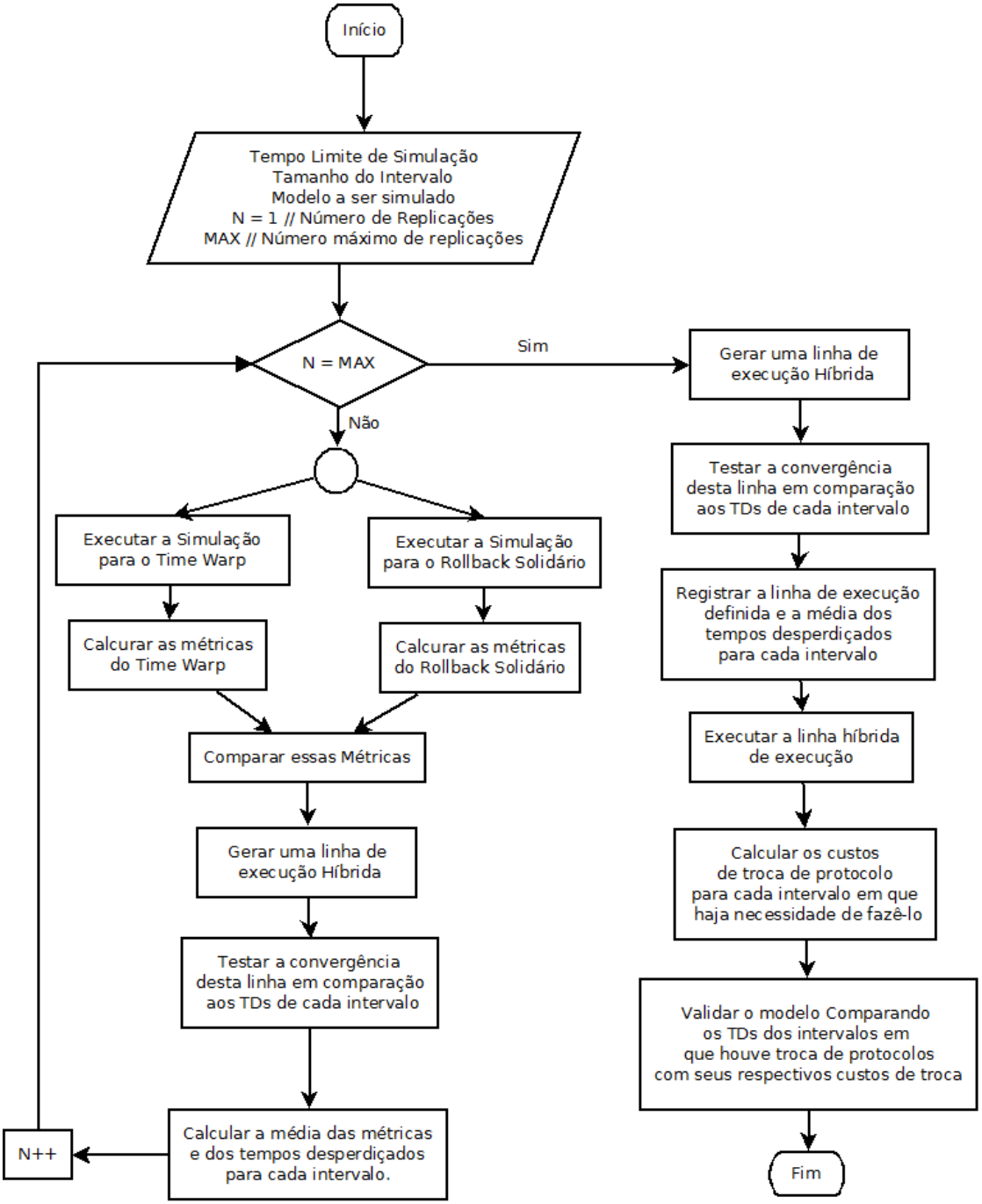

Figura 3.9: Fluxograma do método apresentado

durante estes intervalos. São elas: tempo médio de *rollback*, frequência de *rollbacks*, *rollback* por eventos, eficiência e mensagens de *rollback*.

Para garantir a comparação dos intervalos de simulação e a independência no registro de dados, foi implementado um mecanismo de cálculo de GVT. Essa tática assegura que não haverá interferência no registro de dados de desempenho, de um intervalo para outro.

Na segunda parte, a linha de execução híbrida, gerada na primeira parte, é testada utilizando outros dois parâmetros calculados a cada intervalo: o tempo desperdiçado por computações errôneas (TD), somada ao tempo de cálculo do GVT (TT), e o custo total da mudança de protocolos (CT). Esse teste tem a finalidade de validar as métricas, e resultará em uma segunda linha de execução híbrida que poderá convergir, totalmente ou não, com a linha gerada na primeira parte.

No capítulo seguinte são apresentados alguns detalhes de implementação do mecanismo *Protocol Swapper* e como ele funciona, deixando para o capítulo 5 a apresentação de dados gerados a partir de estudos de casos de alguns modelos de simulação.

4 Modelagem e Especificações de Código

Este capítulo apresenta detalhes de implementação, modelagem e especificações de código do mecanismo *Protocol Swapper* apresentado no capítulo 3. Esta implementação foi desenvolvida a partir de ideias e de algumas partes do *framework* proposto em Azevedo (2012), no que concerne à implementação básica dos protocolos de sincronização otimistas *Time Warp* e *Rollback* Solidário. Foram também acrescidos à implementação, mecanismos para (1) o registro do GVT e (2) e registro de informações relativas ao desempenho.

O gerador de modelos aleatórios apresentado no trabalho de Azevedo (2012), e anteriormente no trabalho de Cruz (2009), foi utilizado nesta implementação e não sofreu alteração. Este algoritmo gera, como dado de saída em um arquivo, uma matriz de adjacência de ordem n (n sendo igual ao número de processos do modelo), representando o grafo que caracteriza o modelo de filas. Os valores desta matriz são preenchidos pelos valores em percentual da probabilidade de envio de eventos entre os processos do modelo. Além dessa matriz, constam no arquivo: o número de processos do modelo, os valores das taxas de nascimento e de morte de eventos para cada processo.

4.1 Tecnologias Utilizadas

A aplicação desenvolvida segue a mesma linha do trabalho de Azevedo (2012), utilizando-se da tecnologia Java e com a versão atualizada da API MPJ-Express.

O motivo de se optar pela continuidade do uso da plataforma Java nesta implementação, deve-se à sua alta portabilidade e compatibilidade entre diferentes ambientes de desenvolvimento. A padronização do seu código fonte permite que a aplicação seja desenvolvida e executada em máquinas com arquiteturas e sistemas operacionais diversos (DEITEL; DEITEL, 2010). Por exemplo, uma aplicação pode ser desenvolvida em uma máquina com o sistema operacional (SO) Windows e executada em um ambiente distribuído, como um *cluster*, com SO Linux, sem que para isso seja necessária nenhuma alteração no código fonte da aplicação. Em contrapartida, aplicações baseadas na linguagem C ou C++ possuem maior agilidade em relação à aplicações em linguagem Java, entretanto, por uma questão de escalabilidade, os ganhos em se utilizar Java superam as linguagens C e C++, inclusive por conta da gama de recursos desta linguagem.

A API MPJ-Express é implementada em Java (SHAFI; MANZOOR, 2009), o que facilita a sua instalação e configuração em relação a outras implementações MPI nativas de sistemas operacionais Linux (AZEVEDO, 2012), especialmente em IDEs (*Integrated Development Environment*) como o NetBeans e o Eclipse (SHAFI; HAMID, 2014). Portanto, essa API não depende da pré-existência de qualquer implementação MPI nativa para sua execução, além do código nativo já empacotado na sua instalação (*shared libraries*). Essa característica permite a execução de uma aplicação distribuída em um ambiente heterogêneo de plataforma, possibilitando que cada processo de uma simulação execute em um SO diferente.

Antes que a simulação se inicie, é necessário que alguns parâmetros sejam definidos como: quantidade de intervalos, tamanho dos intervalos, tempo máximo de simulação, critério de parada da simulação, intervalo de salvamento de estados, intervalo de tempo para o cálculo do GVT, entre outros. Para a entrada de dados é utilizado um arquivo de propriedades (*properties file*), um arquivo sequencial cujas informações consistem em uma chave e seu respectivo valor. Esse arquivo de propriedades é lido pelo processo observador, e suas informações são repassadas, via mensagem, aos processos da simulação antes do início da mesma. Esse recurso da linguagem Java possibilita que as informações da simulação possam ser configuradas de forma prática e estruturada, conferindo alta flexibilidade ao projeto.

Por fim, neste trabalho, optou-se por não utilizar o recurso da "reflexão" (*reflection*) na implementação desta ferramenta, diferentemente da proposta de Azevedo (2012). Embora o uso de reflexão possa conferir mais dinamismo, flexibilidade e escalabilidade à aplicação, seu uso acarreta a desvantagem da perda de desempenho, pois a invocação de um método por reflexão é mais custosa, em termos de tempo, que a invocação diretamente no objeto (GUERRA, 2014). Não obstante, como essa ferramenta visa medir o desempenho e comparar dois protocolos distintos, optou-se por não utilizar um recurso que limitaria o desempenho da mesma, tornando-a mais demorada.

4.2 Modelagem do Sistema

A linguagem Java possibilita que as classes que compõem uma aplicação sejam divididas em pacotes, facilitando a gerência dos componentes do aplicativo e a eventual reutilização de *software*, permitindo que outros programas importem classes de outros pacotes (DEITEL; DEITEL, 2010).

O *framework* que implementa os protocolos, e os mecanismos de troca, é ilustrado na figura 4.1. Nesta figura, as classes que implementam os protocolos *Rollback* Solidário e *Time Warp* estão separadas, respectivamente, nos pacotes ***timewarp*** e ***solidaryrollback***, os quais ambos importam as classes do pacote ***framework***. Este pacote contém classes que possuem implementações genéricas para funcionamento da simulação que são estendidas e implementadas pelas classes do pacote específico de cada protocolo. Essas especificações são detalhadas nas subseções seguintes.

4.2.1 Framework para Implementação dos Protocolos de Simulação

A figura 4.2 mostra o diagrama de classes do pacote *framework* (não foi atribuído nome para esse *framework*). Estas classes constituem o cerne de funcionamento do *framework*, possibilitando às classes dos pacotes específicos estendê-las e estabelecer funcionalidades próprias inerentes ao protocolo que implementam. Nos diagramas seguintes, não são representados os métodos *getters* e *setters*, nem mé-

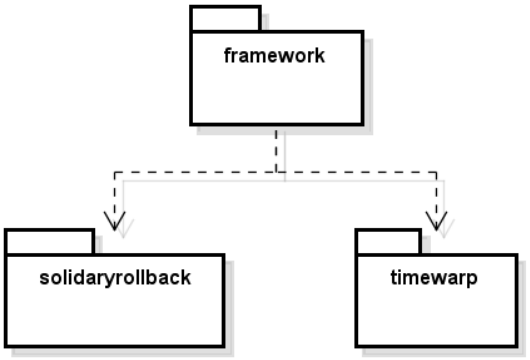

Figura 4.1: Representação esquemática do *framework*

todos que implementam funcionalidades auxiliares e com relevância local restrita à própria classe em que é definido.

Neste diagrama, duas *interfaces* se destacam: **MessageTypes** e **Communication**. A função dessas *interfaces* é estabelecer um padrão de comunicação durante a simulação. A *interface* **Communication** estabelece os métodos que possibilitarão a comunicação entre os processos da simulação, ao passo que a *interface* **MessageTypes** padroniza os rótulos das mensagens utilizadas na ferramenta. Como a ferramenta desenvolvida neste trabalho utiliza a biblioteca de comunicação MPI, os métodos especificados em **Communication** precisam seguir as diretivas do padrão MPI para o envio e recebimento de mensagens. Para tanto, a classe **MPICommunication** implementa a funcionalidade dessa *interface*, aplicando as diretivas do MPI às assinaturas dos métodos definidos.

A classe abstrata **Process** implementa a interface *MessageTypes*, estendendo para todos os processos da simulação a padronização de rótulos para as mensagens que trafegarão na rede. Essa padronização é estendida para todos os processos que participam da execução da simulação distribuída, seja o processo observador, definido pela classe abstrata **ControllerProcess**, ou um processo que represente o modelo simulado, definido pela classe abstrata **SimulationProcess**. Essas classes definem funcionalidades básicas necessárias ao andamento da simulação, uma vez que o mecanismo de cálculo do GVT, desenvolvido para esta ferramenta, requer um processo observador para o seu cálculo, para ambos os protocolos.

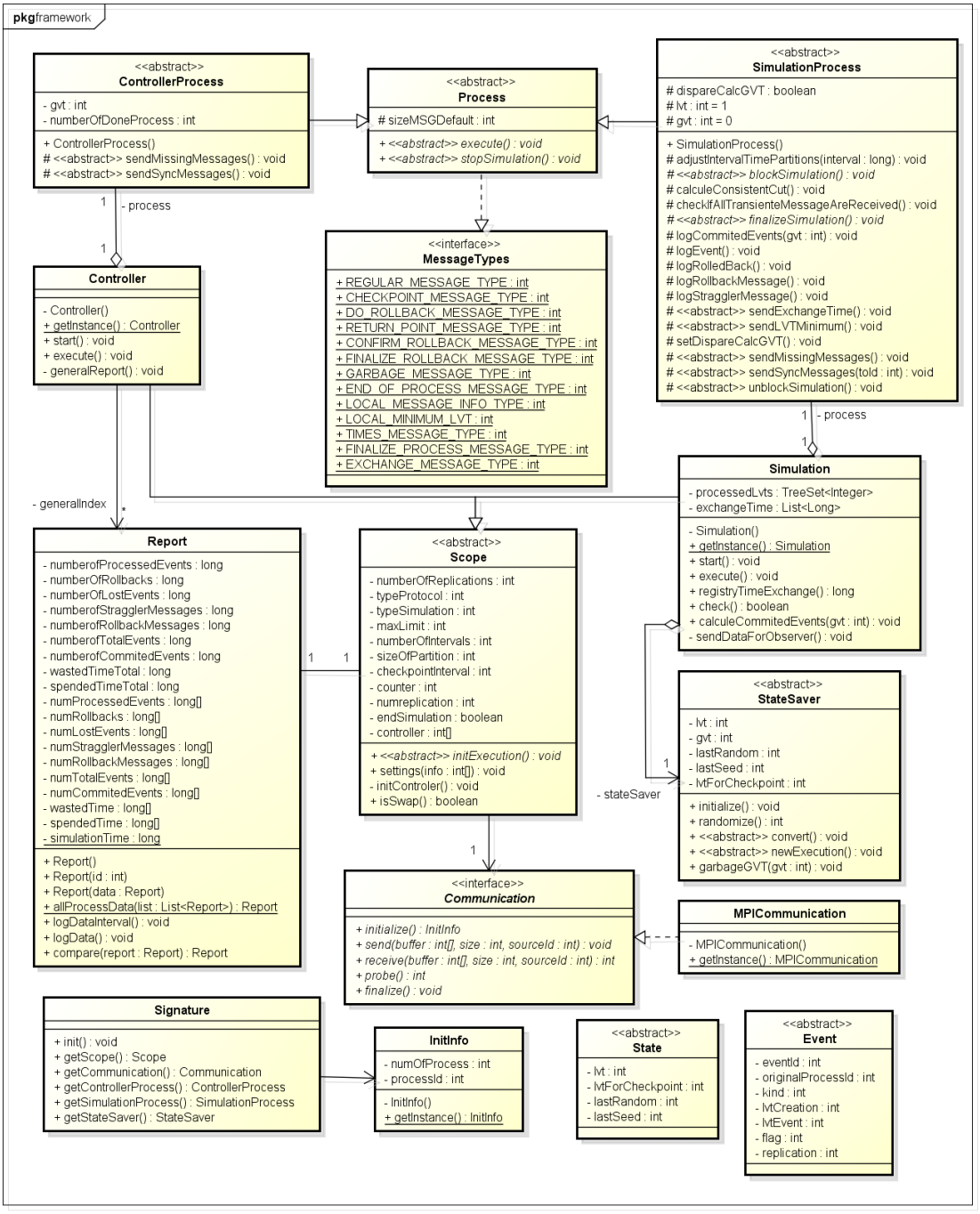

Figura 4.2: Diagrama de classes do pacote *framework*

Como o mecanismo propõe-se a efetuar a troca entre protocolos, as classes **Simulation** e **Controller** foram modeladas para abarcar instâncias de subclas-

ses das classes **SimulationProcess** e **ControllerProcess**, respectivamente, de modo que possam controlar, durante a mudança de intervalos, a troca de protocolos, quando for mais vantajosa. As classes **Simulation** e **Controller** estendem funcionalidades da classe abstrata **Scope**, incluindo o método abstrato **initExecution()**, responsável por inicializar a simulação para os processos da simulação e para o processo observador. A classe **Scope** armazena as informações de inicialização que precisam ser carregadas para cada processo ao início da simulação, ou seja, um vetor de controle que sinaliza qual protocolo deve ser executado em cada intervalo (no caso de uma linha de execução híbrida) e um contador que indica em qual intervalo a simulação se encontra.

A classe **Report** é responsável por registrar os dados de desempenho em cada intervalo de simulação e o total, considerando os dados de todos os intervalos, ao final de sua execução após a chamada dos métodos de registro: **logData()** e **logDataInterval()**. A classe **Report** é associada à classe **Scope**, não obstante, tanto as classes **Simulation** e **Controller** possuem uma instância desta classe, usando-a para anotar os dados de desempenho dos intervalos ou para compilar os dados totais de todos os processos, respectivamente.

Como cada protocolo possui suas particularidades em relação às informações a serem salvas no decorrer da simulação, as classes **StateSaver**, **State** e **Event** foram modeladas como abstratas. Isso permite que as suas subclasses herdem as suas informações básicas e essenciais ao funcionamento da simulação, e adicionem atributos extras inerentes ao funcionamento de cada protocolo. A classe **StateSaver** possui uma referência na classe **Simulation**, e é definida, a priori, para as seguintes finalidades: (1) armazenar a última semente de número aleatório gerada na simulação; (2) disponibilizar um método **randomize()** responsável por gerar novos números aleatórios; (3) definir métodos a serem implementados pelas suas subclasses para a conversão de dados entre protocolos, item explicado nas seções seguintes. As classes que estendem a classe **StateSaver**, para cada protocolo, são responsáveis pelo salvamento de estados da simulação, uma vez que cada protocolo possui suas estruturas de dados próprias para o armazenamento de estados salvos e, consequentemente, dos seus eventos.

Por fim, a classe **Signature** é a única que contém assinaturas de classes dos pa-

cotes específicos de cada protocolo. Esta classe funciona similarmente à uma classe "*factory*", com a diferença que seus métodos não criam novos objetos ao serem invocados, mas apenas retornam o objeto único correspondente à classe em questão. Isso ocorre devido ao fato de algumas classes, tanto do pacote *framework* como dos pacotes específicos, foram construídas de acordo com o padrão de projetos *Singleton* (FREEMAN et al., 2004; GAMMA et al., 1995), por uma questão de economia de memória. Neste padrão, a classe em questão só pode ter um objeto instanciado. As classes que se utilizam deste padrão são: **Simulation**, **Controller**, **MPICommunication**, e as subclasses de **SimulationProcess**, **ControllerProcess** e **StateSaver**. A classe **Signature** também armazena os dados de inicialização, mantendo uma referência para **InitInfo** que contém o número de processos e o *id* do processo.

4.2.2 Time Warp

O pacote ***timewarp*** está representado no diagrama da figura 4.3. Neste diagrama, a classe **TWObserver** implementa as funcionalidades do método **execute()**, herdado da superclasse **ControlleProcess** que, de forma análoga, o herda da sua superclasse **Process**. De modo análogo, a classe **TWProcess** implementa as funcionalidades do método **execute()**, herdado da superclasse **SimulationProcess**, com a diferença de que esta classe comporta uma *thread* apenas para o recebimento e tratamento de mensagens, pois é necessário que esta atividade ocorra em paralelo com o processamento da lista de eventos futuros pelo método **execute()**. A classe **TWProcess** possui ainda uma instância de **TWStateSaver**, responsável pelo salvamento e recuperação de estados em cada processo, além de armazenar a lista atual de eventos futuros. Esses estados, por sua vez, são modelados pela classe **TWState**, que armazena os atributos de sua superclasse **State** e a lista de eventos futuros correspondente àquele estado. Os eventos são representados aqui pela classe **TWEvent**, que estende a classe **Event** e armazena o identificador(*id*) do seu processo criador.

Para que o mecanismo de *rollback* funcione no *Time Warp*, é necessário que haja uma estrutura de antimensagens em cada processo da simulação. A classe

Message representa uma antimensagem no sistema. Esta classe armazena o evento enviado para outro processo e a *id* do processo para onde foi enviado.

Um detalhe importante é que as classes **TWEvent**, **TWState** e **TWStateSaver** possuem um método **convert()**. Esses métodos são necessários durante a transição do protocolo *Time Warp* para o protocolo *Rollback* Solidário. Essas classes, por conterem informações que devem persistir na mudança de intervalos, precisam de um modo de passar suas informações para estruturas específicas do outro protocolo.

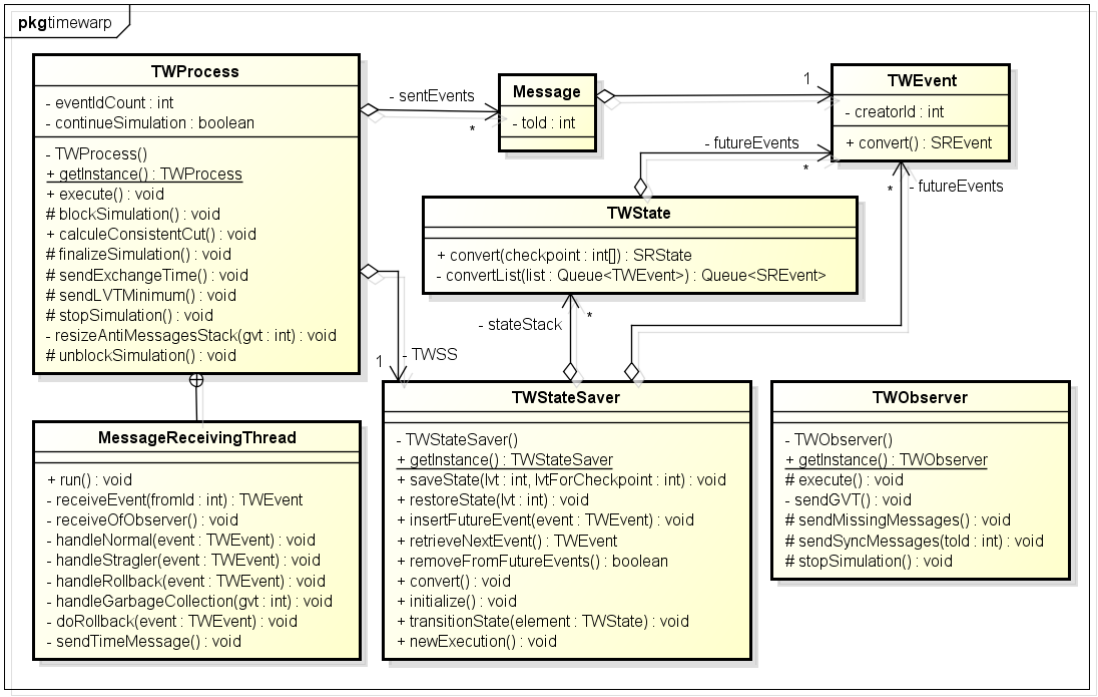

Figura 4.3: Diagrama de classes do pacote *timewarp*

4.2.3 Rollback Solidário

O pacote ***solidaryrollback*** está representado no diagrama da figura 4.4. A implementação deste protocolo ocorre de forma análoga à implementação do **Time Warp**, de modo que suas classes **SREvent**, **SRState** e **SRStateSaver** também possuem métodos de conversão para estruturas correspondentes no protocolo *Time*

Warp. A classe **SRObserver** e **SRProcess**, de maneira análoga à implementação do *Time Warp*, estendem, respectivamente, as funcionalidades das classes **ControlleProcess** e **SimulationProcess**, e implementam seus métodos **execute()**. A classe **SRProcess**, assim como a classe **TWProcess**, possui uma *thread* para recebimento e tratamento de mensagens, que executa paralelamente com o método **execute()**.

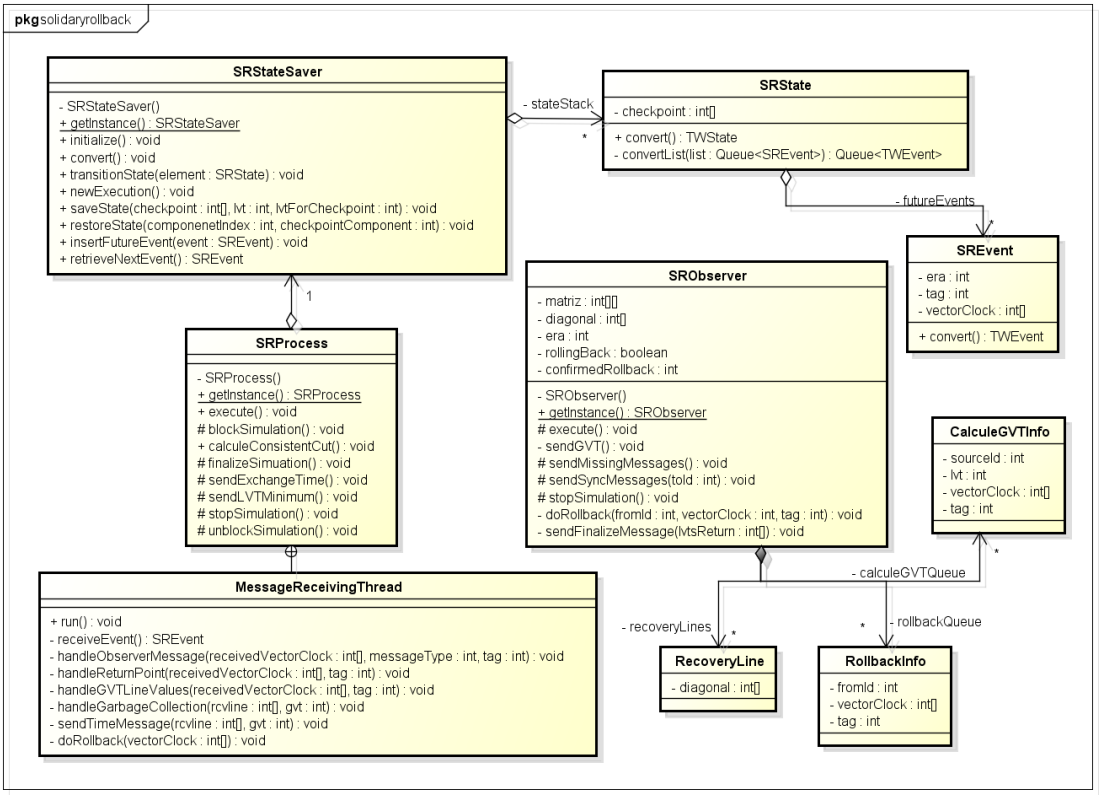

Figura 4.4: Diagrama de classes do pacote *solidaryrollback*

A maior diferença entre os dois protocolos está na forma como efetuam o procedimento de *rollback*. O protocolo *Rollback* Solidário não necessita de anti-mensagens, portanto não precisa armazenar uma lista delas. Isso significa que sua implementação não requer uma classe **Message** como na implementação do **Time Warp**, entretanto, o procedimento de *rollback* no *Rollback* Solidário requer uma maior participação do processo observador. A classe **TWObserver** possui listas para três outras classes: **RecoveryLine**, **RollbackInfo** e **CalculeGVTInfo**. A classe **RecoveryLine** armazena uma linha de recuperação, extraído da diagonal

principal da matriz de *checkpoints*. A classe **RollbackInfo** é usada para "agendar" *rollbacks*, como forma de evitar *rollbacks* simultâneos, desta forma um *rollback* agendado só é executado quando a operação de *rollback* anterior já tiver finalizada. De forma análoga, a classe **CalculeGVTInfo** é usada para "agendar" solicitações de cálculo de GVT. As solicitações agendadas só são efetuadas após o término da última operação de *rollback* ser finalizada.

4.2.4 Relação entre os pacotes

As relações entre as classes do pacote *framework* e os pacotes *timewarp* e *solidaryrollback* são dadas na figura 4.5. As classes **State** e **Event** são mostradas desligadas na figura 4.2, mas são estendidas para os pacotes específicos de cada protocolo. As subclasses destas classes definem os dados a serem utilizados para cada protocolo. As classes **StateSaver**, **SimulationProcess** e **ControllerProcess** também são estendidas aos pacotes específicos.

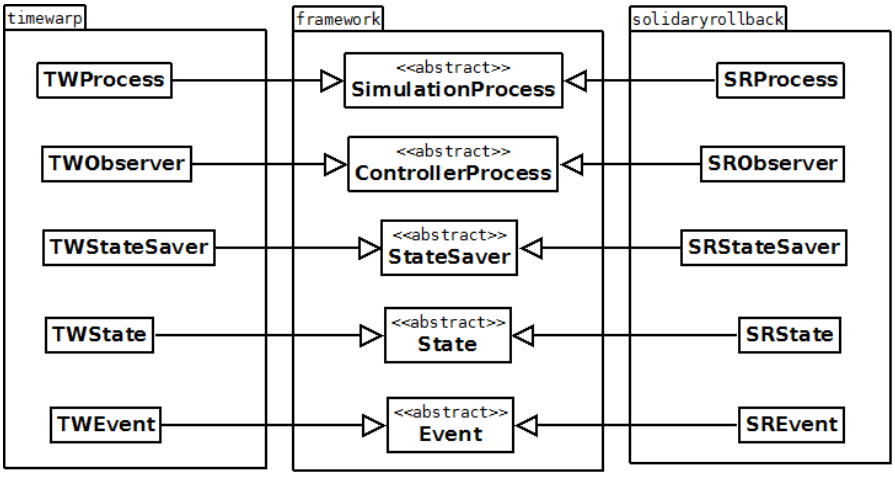

Figura 4.5: Relação entre as classes dos três pacotes do *framework*

4.2.5 Mecanismo de Trocas

O mecanismo de troca de protocolos é esquematizado nos diagramas de sequência das figuras 4.6 e 4.7, para os processos da simulação e o processo observador,

respectivamente. As trocas ilustradas nestas figuras decorrem do *Time Warp* para o *Rollback* Solidário.

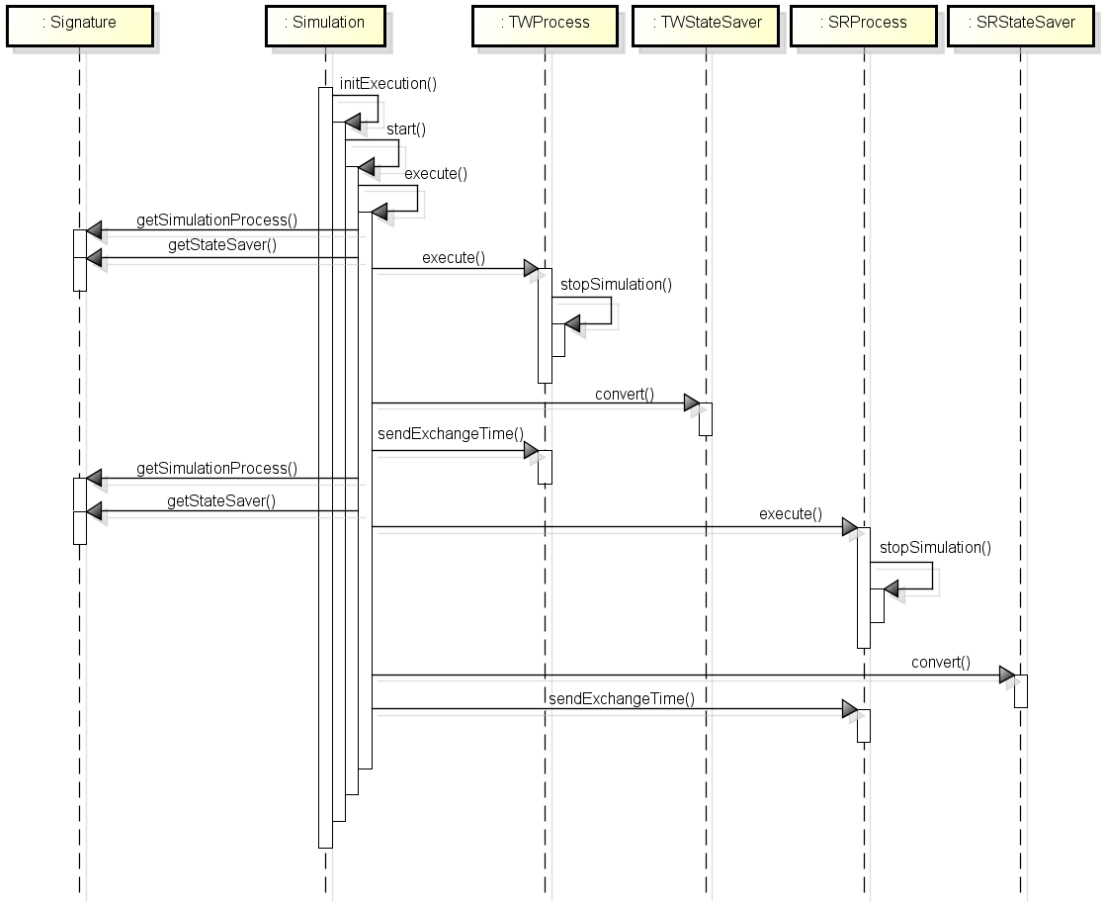

Figura 4.6: Troca de protocolos para um processo de simulação

No caso do processo da simulação, a classe **Simulation** executa o método **initExecution()**, que invoca o método **start()**. Este método tem por finalidade carregar os dados de entrada, recebidos do processo observador via mensagem, no caso o modelo a ser executado, e os dados de inicialização especificados na classe **Scope**. O método **execute()** é chamado pelo método **start()** após a inicialização destes dados pelo método **settings()** da superclasse **Scope**. O método **execute()** consiste em dois *loopings* encadeados: o primeiro, para executar toda a quantidade de replicações especificadas a priori; o segundo, para executar todos os subintervalos de simulação especificados.

No início das iterações do método **execute()**, a classe **Simulation** é informada pelos métodos da classe **Signature**, sobre quais instâncias de subclasses estão sendo utilizadas para o processo de simulação (**getSimulationProcess()**) e para o salvador de estados (**getStateSaver()**). Uma vez na posse dessas instâncias, o método **execute()** de **SimulationProcess** é executado pela respectiva subclasse desta. Esse método inicia a simulação para um processo, até que seja determinada uma troca de protocolos (ou o fim da simulação, caso não haja trocas). Quando isso acontece, o método **stopSimulation()** interrompe a simulação e encerra o método **execute()**.

Após o término de **execute()**, o método **convert()** realiza a conversão dos dados que devem persistir durante a mudança de protocolos, para o salvador de estados correspondente do protocolo a ser executado. Estes dados a serem preservados são: (1) a semente de números aleatórios; (2) a lista de eventos futuros, (3) o valor do GVT e (4) a linha de LVTs atuais. Após a conversão das estruturas de dados, é enviado ao processo observador, pelo método **sendExchangeTime()**, o tempo decorrido da troca de protocolos.

De maneira análoga, o diagrama da figura 4.7 representa esquematicamente o funcionamento do mecanismo de troca de protocolos para o processo observador. A classe **Controller** executa o método **initExecution()**, que invoca o método **start()**. Este método tem por finalidade carregar os dados de entrada, no caso o modelo a ser executado, e os dados de inicialização especificados na classe **Scope**. Esses dados de entrada são carregados a partir do arquivo de propriedades e do arquivo que contém o modelo a ser simulado. Posteriormente, estes dados são enviados aos processos da simulação via mensagem.

O método **execute()** da classe **Controller** funciona de forma similar ao método **execute()** da classe **Simulation**, com a diferença de que o método chamado em **Signature** é **getControllerProcess()**, que retorna a instância da subclasse utilizada como processo observador naquele momento. Além de que, a classe **Controller** não requer nenhum mecanismo de salvamento de estados. Os estados globais atualizados, gerados a partir do cálculo do GVT, durante a mudança de intervalos, são armazenados em um arquivo a parte pelo próprio processo observador, pois, uma vez que não se poderá retornar a eles durante a simulação, não há neces-

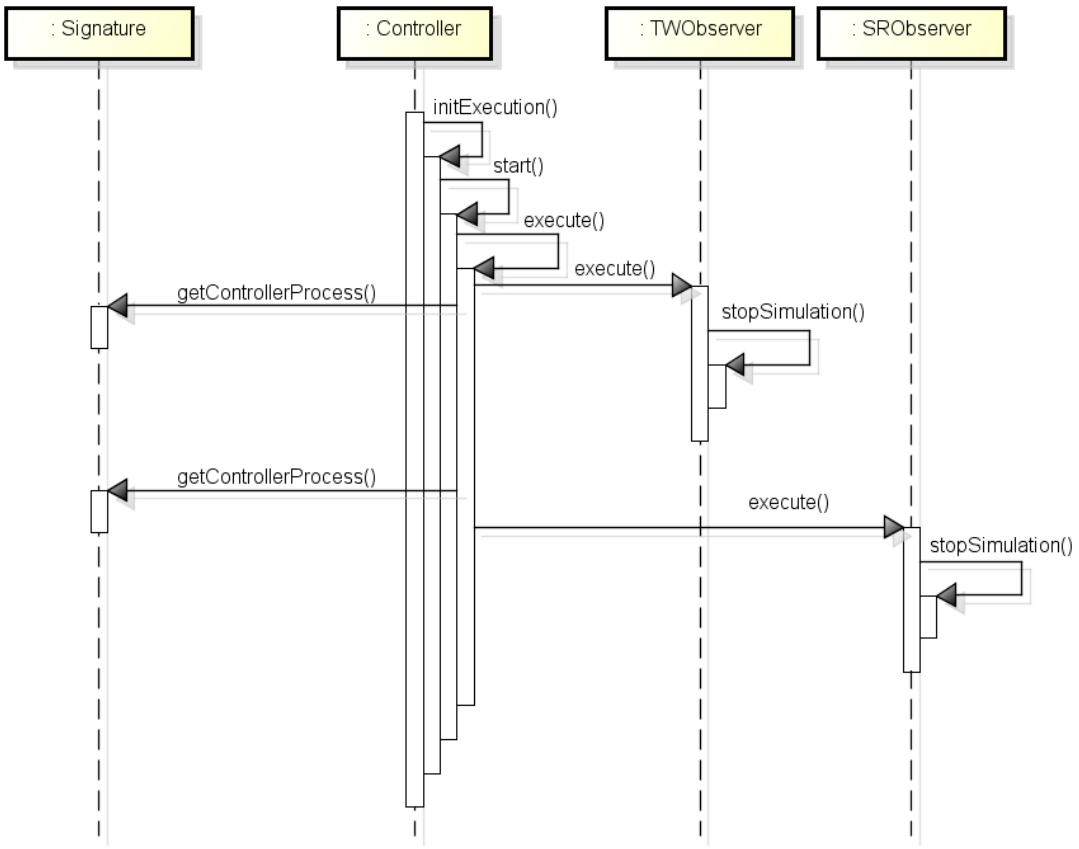

Figura 4.7: Troca de protocolos para o processo observador

sidade de lhes dispensar uma estrutura para armazená-los na memória. O método **stopSimulation()** interrompe a simulação no processo observador, recomeçando o ciclo para as demais trocas de protocolos até o final da simulação.

A indicação sobre quais transições de intervalo se deve efetuar a mudança de protocolos é feita pelo *array* **controller** da classe **Scope**. Esse *array* tem o tamanho igual ao número de intervalos da simulação, e especifica qual protocolo deve ser executado em cada um dos intervalos correspondentes. Essa informação é carregada ao início da segunda etapa do método, no qual as linhas de execução híbridas são testadas. As mudanças de protocolos ocorrem em momentos de transição entre um intervalo e outro, nos quais são calculados o GVT e os respectivos valores de LVT de cada processo, os quais a simulação deve retomar seu processamento com outro protocolo.

4.3 Mecanismos de Cálculo do GVT

Ao final de cada intervalo de simulação, é efetuado o cálculo de GVT para determinar o progresso da simulação e realizar uma espécie de *garbage collection*, eliminando das filas registros com valor de LVT inferior ao GVT calculado. A operação de cálculo do GVT ocorre à cada transição de intervalos, independente se haverá troca ou não de protocolos, ou seja, antes da operação *stopSimulation()* nos diagramas anteriores.

4.3.1 Time Warp

O cálculo do GVT no *Time Warp* é realizado pela implementação do algoritmo de Mattern, às suas classes internas. A figura 4.8 ilustra como se dá esse cálculo ao final de um intervalo que esteja executando o protocolo *Time Warp*.

Após a inicialização dos métodos **execute()** dos processos de simulação, o método **check()** é invocado à cada interação para verificar se a simulação atinge as condições adequadas para efetuar o cálculo do GVT. Se essas condições forem atingidas, o método **check()** invoca o método **calculeConsistentCut()**, que inicia o cálculo do GVT. Neste método, é enviado ao processo observador o número de mensagens enviadas a outros processos e o número de mensagens recebidas; em seguida o processamento da simulação (o processamento da lista de eventos futuros e a verificação da chegada de novos eventos ao sistema) é bloqueada. Após receber a lista de mensagens enviadas e recebidas por todos os processos da simulação, o processo observador retorna à cada processo da simulação a quantidade de mensagens transientes a serem recebidas. Após todas serem recebidas, o método **sendLVTMinimum()** envia para o processo observador o menor valor do LVT calculado, entre o LVT de criação se suas "mensagens vermelhas", e o seu LVT atual. O processo observador calcula o GVT como o menor valor entre os valores de GVT recebidos, e os envia aos processos da simulação. O método **handleGarbageCollection()** utiliza o valor de relógio global recebido para fazer uma operação de "garbage collection", eliminando da estrutura de antimensagens (**resizeAntiMessagesStack()**) e da pilha de estados salvos (**garbageGVT()**),

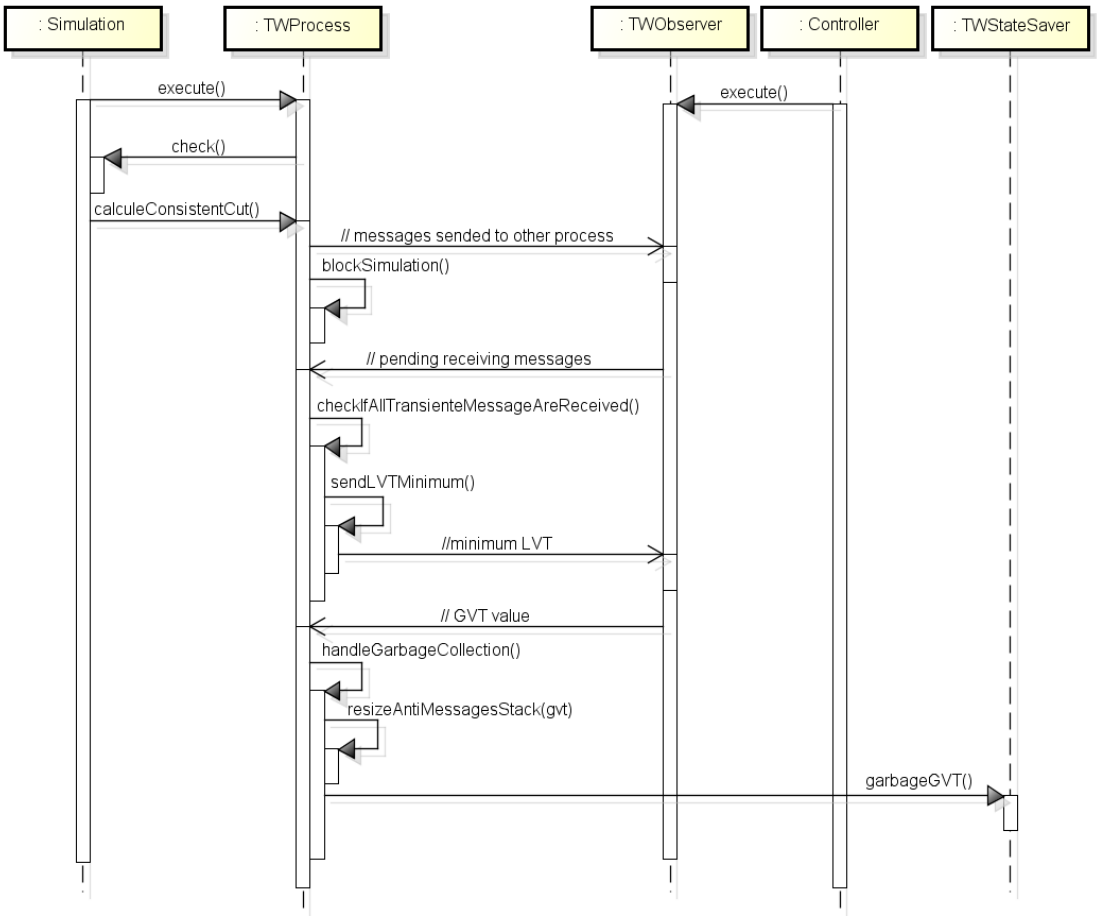

Figura 4.8: Cálculo do GVT para o *Time Warp*

os registros que possuem o valor de LVT inferiores ao GVT recebido. Por fim, a simulação é desbloqueada pelo método (**unblockSimulation()**), dando início ao próximo intervalo de simulação.

4.3.2 Rollback Solidário

De modo análogo, o cálculo de GVT no *Rollback* Solidário é realizado por uma variação do algoritmo de Mattern que abarque suas particularidades, que envolvem a participação do processo observador na operação de *rollback*. Essas particularidades residem no fato dos procedimentos de *rollback* solidário requererem a participação do processo observador. Isso exige que todas as operações de

rollback iniciadas antes, ou durante o cálculo do GVT, sejam completadas antes que esse cálculo seja completado. Para tanto, o processo observador se utiliza de uma fila de solicitações de pedidos de cálculo de GVT, usando uma fila de instância da classe **CalculeGVTInfo**. A figura 4.9 ilustra como se dá esse cálculo ao final de um intervalo que esteja executando o protocolo *Rollback* Solidário, elucidando onde as alterações são realizadas.

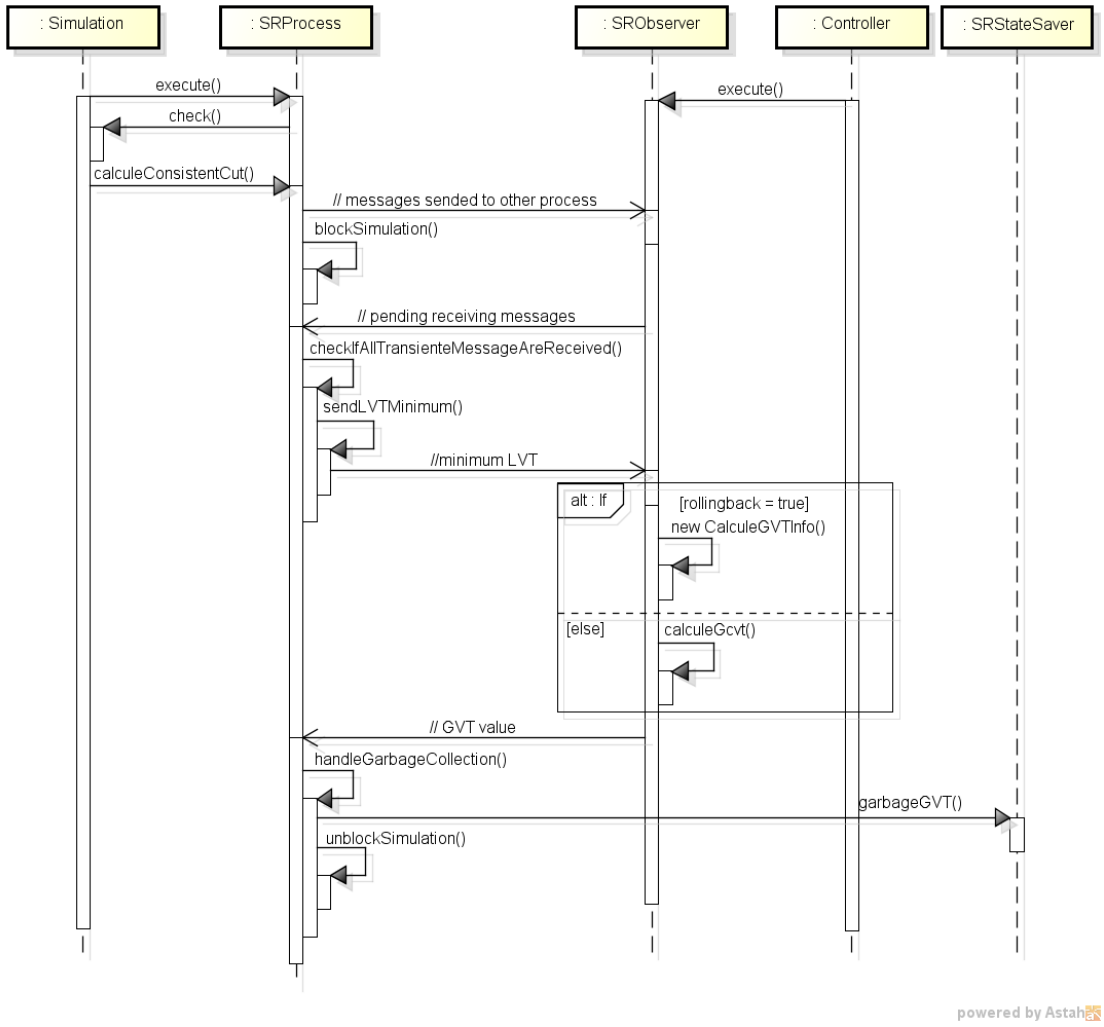

Figura 4.9: Cálculo do GVT para o *Rollback* Solidário

4.4 Outras Otimizações

Além dos mecanismos descritos nas seções anteriores, uma otimização importante para o bom funcionamento do *Protocol Swapper* foi a redução de instanciações de objetos à cada iteração da simulação.

O código de Azevedo (2012) implementa a instanciação de objetos com escopo de método para o tratamento de eventos, quanto ao envio e recebimento de mensagens interprocessos. Isso aumenta demasiadamente o tamanho do *heap* do Java quando a simulação precisa ser executada por longos períodos de tempo.

Uma solução adequada a esse problema, adotada pela implementação deste trabalho, foi o uso prévio de algumas variáveis fixas com escopo de classe, de modo a tornar a instanciação excessiva de variáveis locais desnecessária. Isso possibilita uma melhoria de desempenho com o aumento da reutilização física de memória, usando uma mesma estrutura de memória para os dados gerados a cada iteração.

A dificuldade que essa abordagem impõe, é quanto à sincronização no uso desses variáveis. Um atributo com escopo de classe pode ser usado por mais de um método, e quando se trata de um ambiente *multi-threading*, há certos cuidados que devem ser tomados para que resutados insconsistentes não sejam gerados por falha de sincronização e comprometam a corretude do programa de simulação. Para tanto, a preocupação com a sincronização no uso dessas variáveis por métodos de *threads* distintas e especificá-las em número correto para tratar o problema com uma razoabilidade adequada, requiseram reestruturações em partes do código que tratem da implementação específica dos protocolos do código legado do *framework* de Azevedo (2012).

Essas otimizações, embora vitais para o bom funcionamento e desempenho do *Protocol Swapper*, mantiveram intacta a estrutura de relacionamento entre classes da proposta apresentada anteriormente. Portanto, neste trabalho não foram detalhadas otimizações laterais à sua proposta central, além do que foi mencionado nesta seção.

4.5 Considerações Finais

Neste capítulo foram expostos alguns detalhes de implementação do mecanismo *Protocol Swapper*, proposto no terceiro capítulo deste trabalho. A utilização da linguagem Java permite a modularização do *software* em pacotes, o que facilita sua expansão e reutilização de código. O *framework* é dividido em três pacotes, um com implementações para o *Time Warp*, outro para o *Rollback* Solidário e um para uso geral de funcionalidades que são estendidas para os pacotes específicos. Também foram apresentados detalhes acerca do funcionamento do mecanismo de troca de protocolos e dos mecanismos de cálculo de GVT, para ambos os protocolos.

A ferramenta proposta neste capítulo agrega, e torna operacional, as implementações basilares dos protocolos em Azevedo (2012). As novas funcionalidades adicionadas ao código base, neste trabalho, possibilitaram a concepção e a implementação do método de comparação de protocolos otimistas. Esses algoritmos possuem as seguintes finalidades:

- divisão da simulação em subintervalos: a subdivisão prévia da simulação em intervalos menores, possibilita a análise da simulação dentro de janelas específicas de tempo, em recortes delimitados por cortes consistentes. Isso cria condições para a comparação de desempenho e troca de protocolos nas transições de intervalos;

- cálculo de GVT: o algoritmo de Mattern, e sua adaptação para o *Rollback* Solidário, são implementados, cumprindo, deste modo, duas finalidades distintas: (1) controlar os tamanhos das listas (*garbage collection*), consequentemente otimizando o uso da memória (desta forma, evitando ocorrência de *overflow*), e (2) possibilitar que cada intervalo seja consistente em si mesmo, ou seja, que os dados de desempenho de um intervalo não interfira, ou sofra interferências, dos seus intervalos adjacentes;

- geração de relatórios: ao final da simulação, os dados de desempenho de todos os processos do modelo simulado são enviados por meio de mensagem ao processo observador, para que sejam gerados os relatórios finais, de todos os processos, em apenas uma máquina. O levantamento das informações de

desempenho, e o cálculo de métricas, são efetuados, para todos os intervalos, em cada processo. Ao final da simulação, esses dados são compilados e enviados ao processo observador, de modo que é, também, calculada a média dos resultados, na elaboração de um relatório com os resultados totais, considerando os dados de todos os processos;

- troca de protocolos: a divisão da simulação em intervalos possibilita que, ao final de cada subintervalo, o algoritmo de cálculo de GVT seja acionado. Isso possibilita que a simulação possa ser reiniciada, nas mesmas condições em que parou no intervalo anterior, com um protocolo distinto do protocolo do intervalo anterior, se assim for estabelecida a necessidade da troca de protocolos.

O uso do padrão de projeto *Singleton* também contribuiu no desempenho do *Protocol Swapper* quanto à economia de memória. Este padrão reduz a necessidade de se criar várias instâncias de uma classe, limitando-as à apenas uma por classe durante todas as replicações da simulação de um modelo. Isso acarreta a vantagem da não vinculação do número de instâncias de boa parte dos objetos do *framework*, ao número de iterações, ou ao número de replicações, da simulação.

Além das funcionalidades descritas, a implementação deste trabalho visou atenuar a dificuldade em relação ao crescimento do *heap* do Java, devido às instanciações repetitivas de objetos, que se davam à cada iteração da simulação. Esse problema foi resolvido mediante à instanciação prévia de algumas variáveis fixas, e da especificação de *buffers* de tamanho fixo, com escopo de classe, que pudessem ser usados no tratamento de eventos, envio e recebimento de mensagens, durante a simulação. Para que essa dificuldade fosse resolvida de modo razoável em um ambiente *multi-threading* de um processo, foram necessárias reestruturações no que diz respeito à sincronização de métodos no acesso às essas variáveis.

O capítulo seguinte trata a respeito dos testes realizados com o *framework Protocol Swapper* para diferentes tipos de modelos.

5 Experimentos e Resultados

Este capítulo objetiva apresentar e discutir os resultados dos testes realizados com a ferramenta apresentada no capítulo anterior. Realizados em um *cluster* com 11 máquinas, no laboratório GPESC-IESTI UNIFEI, estes testes coletaram os dados de desempenho de diversos modelos simulados. Os experimentos foram realizados seguindo-se as diretivas definidas no capítulo 3, divididas em 2 etapas, as quais podem ser divididas, resumidamente, em:

- 1^a. etapa:
    1. Execução da simulação dos modelos utilizando o protocolo *Time Warp*, em N replicações;
    2. Execução da simulação dos modelos utilizando o protocolo *Rollback* Solidário, em N replicações;
    3. Cálculo das linhas de execução médias para cada protocolo e suas linhas de tempo desperdiçado correspondente;
    4. Validação e análise de convergência dessas linhas em relação ao tempo desperdiçado;
    5. Comparação entre as linhas de execução dos dois protocolos e a determinação de uma linha de execução híbrida.
- 2^a. etapa:
    1. Teste da linha de execução híbrida obtida e cálculo dos custos de troca de protocolos;

2. Validação e análise de convergência da linha de execução híbrida em relação ao custo de troca entre protocolos.

As seções seguintes mostram o detalhamento dos experimentos realizados e suas respectivas conclusões.

5.1 Experimentos Preliminares

Testes preliminares realizados em laboratório, com ambos os protocolos, comprovaram que a simulação distribuída obtém vantagens de desempenho sobre a simulação executada de forma sequencial. Os gráficos das figuras 5.1, 5.2, 5.3 e 5.4 ilustram os dados de *speedup* e da eficiência, das simulações de modelos de 4, 50 e 100 processos, executadas em ambiente sequencial e com configurações com 2, 3, 4 até 5 máquinas com a seguinte configuração: Core 2 Duo CPU Q6600 @ 2.40GHz (L2, 4Mb), 4Gb DDR III, Rede Ethernet 10/100 com sistema operacional Linux. Estas duas métricas, *speedup* e eficiência, são definidas, respectivamente, como a divisão do tempo gasto pela simulação no ambiente sequencial (T_{seq}) pelo tempo gasto pela simulação paralela (T_{par}), e a divisão do *speedup* pelo número de processadores (TANG; LEE; HE, 2012). As equações 5.1 e 5.2 representam essas definições em termos matemáticos para *speedup* e eficiência, respectivamente.

$$\text{Speedup} = \frac{T_{seq}}{T_{par}} \tag{5.1}$$

$$\text{Eficiência (Speedup)} = \frac{Speedup}{\text{nº de processadores}} \tag{5.2}$$

Os modelos utilizados nesses experimentos são redes de filas simples, que não executam nenhuma operação além do cálculo do seu respectivo LVT à cada processamento de um evento. O modelo de 4 processos foi definido à priori, ao passo que os modelos de 50 e 100, foram gerados por meio do gerador aleatório mencionado no capítulo 4. Esses tipos de modelos, chamados de "genéricos", neste trabalho e em Azevedo (2012), serão mais bem explicados na seção seguinte.

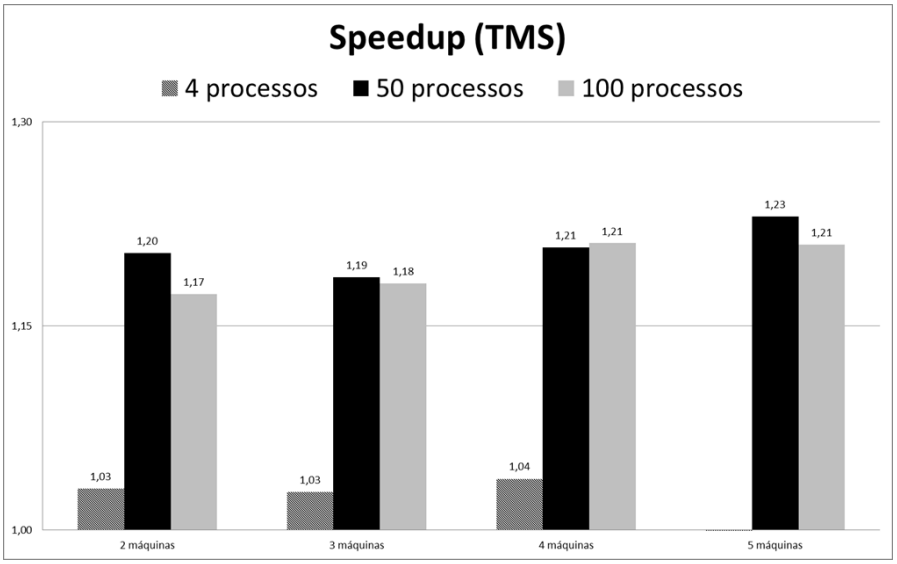

Figura 5.1: Gráfico dos testes preliminares: *speedup* (TMS)

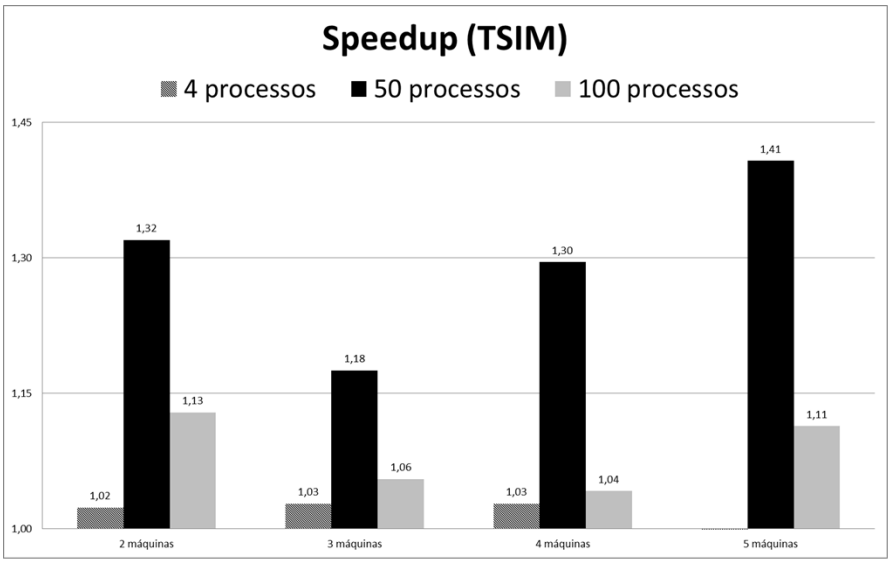

Figura 5.2: Gráfico dos testes preliminares: *speedup* (TSIM)

As siglas TMS e TSIM significam, respectivamente, o tempo médio da simulação e o tempo máximo da simulação, ou seja, a quantidade de tempo que o último processo levou para para atingir o seu objetivo. Os gráficos das figuras 5.1 e 5.2 mostram a envolução do *speedup* para o TMS e para o TSIM, respectivamente. Esses gráficos demonstram que, apesar do *speedup* oscilar e cair de valor com o

aumento do número de máquinas em alguns casos, a simulação paralela aumenta o desempenho da simulação em relação à simulação sequencial.

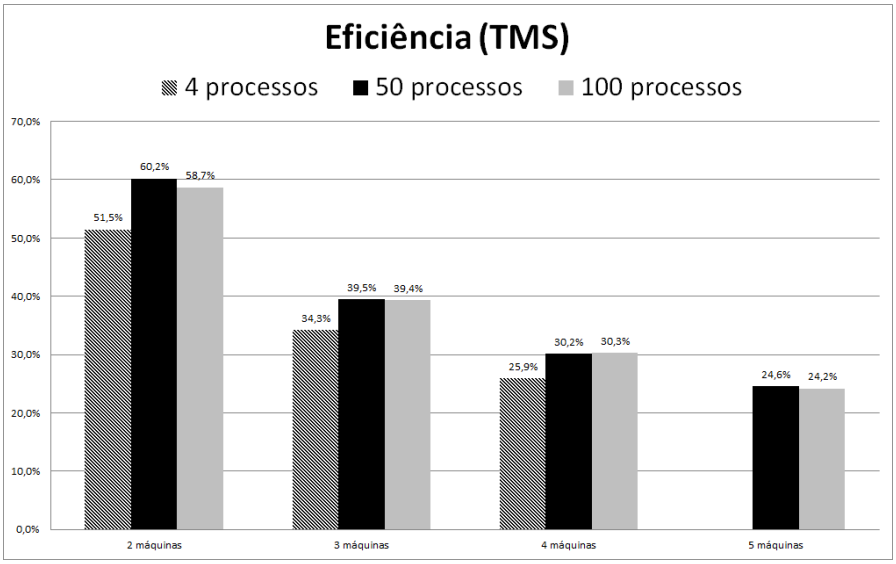

Figura 5.3: Gráfico dos testes preliminares: eficiência (TMS)

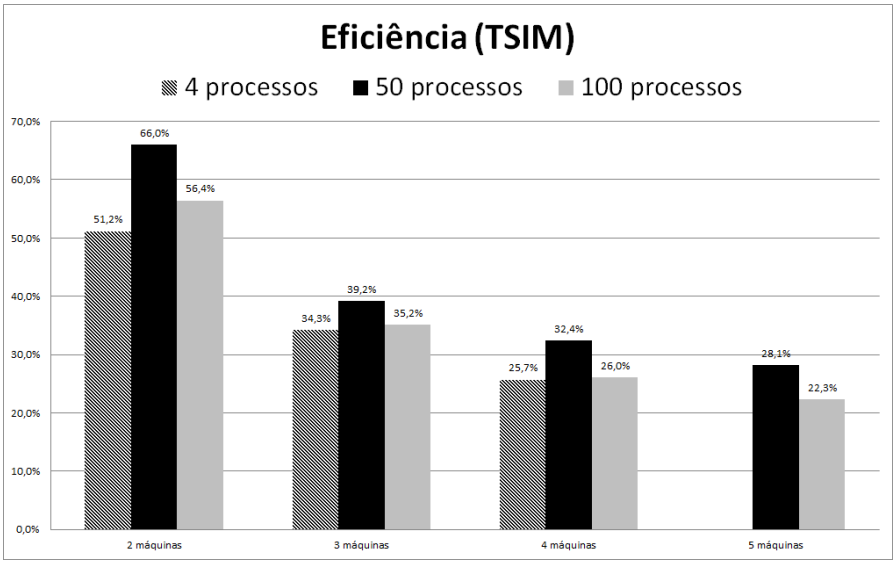

Figura 5.4: Gráfico dos testes preliminares: eficiência (TSIM)

Os gráficos das figuras 5.3 e 5.4 demonstram que, apesar do número de processadores aumentar, o aumento do desempenho da simulação não ocorrerá, necessariamente, na mesma proporção. Isso se deve ao fato de que o *overhead* de

comunicação entre as máquinas em uma simulação paralela diminui em grande parte o ganho de desempenho com a simulação sequencial. O fato do programa de simulação pararela executado seguir o padrão SRIP (*Single Replication in Parallel*) contribui significativamente para o aumento desse *overhead*.

Estes experimentos são melhor detalhados em NUNES et al. (2015).

5.2 Modelos Utilizados

Os modelos definidos em Azevedo (2012), como base de teste, são utilizados na análise metodológica do método proposto por este trabalho, tanto os modelos menores: **M1** (4 processos), **M2** (10 processos), **M3** (20 processos); quanto os modelos maiores com 50 processos: **G1**, **G2** e **G3**. Além desses, foram gerados, para efeito de comparação neste trabalho, outros 3 modelos com 100 processos cada, sendo eles **S1** (10 processos de entrada e 50 processos de saída), **S2** (10 processos de entrada e 60 processos de saída) e **S3** (10 processos de entrada e 70 processos de saída), criados pelo gerador de modelos aleatórios. Esses modelos representam sistemas de redes de filas no padrão M/M/1, ou seja, cada fila dispõe de apenas um canal de atendimento (MORAES; SILVA; REZENDE, 2011).

Como o objetivo deste trabalho é comparar os dois protocolos otimistas apresentados, não foram levantados os dados inerentes aos modelos simulados, ou seja, os dados de saída gerados a partir do modelo particular que está sendo simulado. Isso foge ao escopo deste trabalho, inclusive, porque os modelos gerados, apesar de ponderados e validados, não executam nenhuma operação significativa além do cálculo do seu LVT, com base na geração de números aleatórios. Esses modelos genéricos são apenas para testar os protocolos avaliados.

As figuras 5.5, 5.6 e 5.7 mostram, respectivamente, os modelos **M1**, **M2** e **M3**. Esses modelos foram definidos à priori no trabalho de Azevedo (2012) arbitrariamente, e estão representados na forma de diagramas de estado, onde cada estado representa um processo com uma fila e suas arestas a probabilidade de envio de eventos de um processo à outro.

Também é ilustrado as respectivas probabilidades de entrada de novos eventos

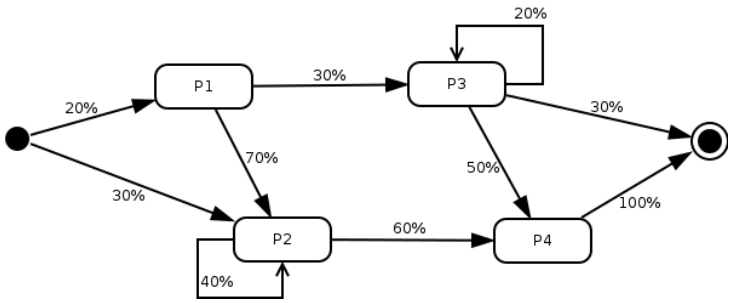

Figura 5.5: M1: Modelo com 4 processos (AZEVEDO, 2012)

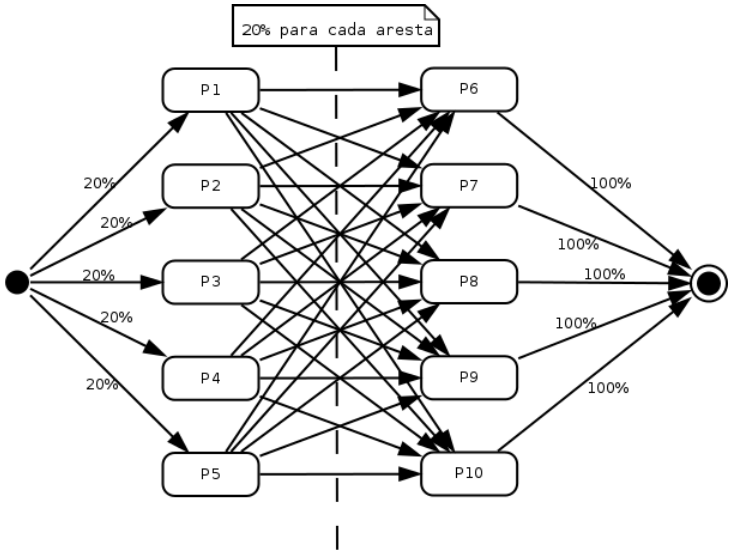

Figura 5.6: M2: Modelo com 10 processos (AZEVEDO, 2012)

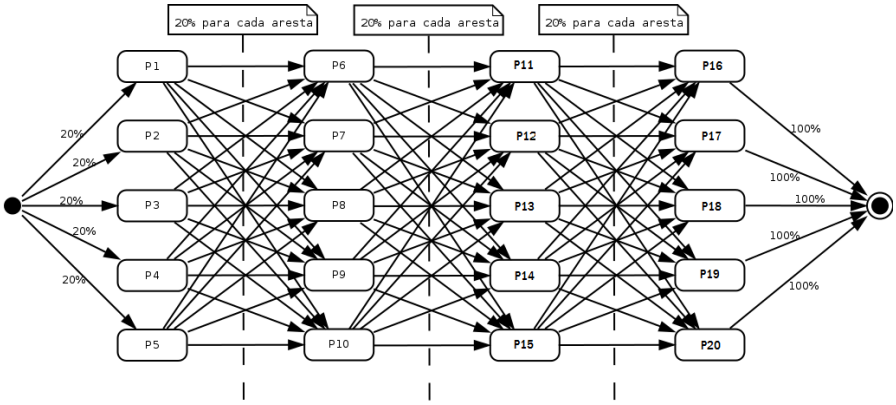

Figura 5.7: M3: Modelo com 20 processos (AZEVEDO, 2012)

(taxa de nascimento ou taxa de chegada) e saída de eventos (taxa de morte ou taxa de saída) do sistema. As taxas de chegada e saída são representadas em cada diagrama, respectivamente, pela porcentagens das aresta que saem dos seus estados iniciais e pelas porcentagens das arestas que chegam aos seus estados finais.

5.3 Configuração das Máquinas

Os experimentos centrais deste trabalho foram realizados sobre um *cluster* homogêneo de 11 máquinas, com uma máquina operando em *front-end* e com as 10 máquinas restantes operando em *back-end*, no laboratório GPESC-IESTI UNIFEI. A configuração dessas máquinas constam a seguir:

```
Core(TM) 2 Quad CPU Q8200 @ 2.33GHz (L2, 4Mb)
Memória RAM: 4 Gb DDR III
Placa de Rede: 82567LM-3 Gigabit Network Connection Ethernet 10/100
Sistema Operacional: Linux Ubuntu 14
```

5.4 Análise de Resultados

Para a realização destes experimentos, foi fixado um tempo total de 10 minutos, divididos em intervalos de 1 minuto, para cada replicação da simulação. Foi fixado, também, o número de 10 replicações a serem executadas para cada protocolo em cada modelo. Com base nesses dados de entrada, foram executados experimentos para os modelos listados com a configuração máxima disponível, ou seja, utilizando as 11 máquinas disponíveis no *cluster* (exceto para o modelo M1 com quatro processos, que usará 5 máquinas - para os 4 processos mais o processo observador). Os experimentos visaram, sobretudo, explorar o máximo de paralelismo possível na execução da simulação para todos os modelos definidos.

5.4.1 Metodologia

Basicamente, ao final de cada replicação de uma simulação distribuída executada para um modelo, o sistema gera dois tipos de relatórios para cada processo da simulação: um relatório contabilizando os dados de desempenho para cada intervalo e, outro, os dados para o tempo total simulado. Em cada um desses relatórios, constará os valores de desempenho, e o cálculo das métricas, dividido por intervalo e com os dados totais da simulação. Vale recapitular que as métricas definidas no capítulo 3 para a aferição e comparação de desempenho dos protocolos são:

- TMR – Tamanho Médio de *Rollback*
- FR – Frequência de *Rollback*
- RE – *Rollbacks* por Eventos
- Eficiência
- Mensagens de *Rollback*

Após todos os processos da simulação enviarem seus dados para o processo observador, o mesmo calculará dois relatórios gerais (dividido por intervalo e tempo total simulado), nos quais o valor das métricas será calculado a partir da soma dos dados de desempenho de todos os processos (exceto, os tempos desperdiçados, dos quais são tirados uma média). A partir de então, obtem-se relatórios com os valores gerais da simulação (de todos os processos da simulação), divididos por intervalos de tempo e por tempo total simulado. Ao final da simulação, obtém-se resultados nos moldes da tabela 5.1. Nessa tabela, são apresentados os valores das métricas, e dos tempos desperdiçados (TD), calculadas com base nos dados por intervalo dos resultados de todos os processos.

Ao final de 10 replicações, 10 tabelas desse tipo são geradas (a tabela é uma abstração, pois, o que é gerado, de fato, é um relatório textual com essas e outras informações) com dados de cada replicação para o *Time Warp*. Consequentemente, mais 10 são geradas com o *Rollback* Solidário, para este modelo. A compilação

Modelo M1 - TIME WARP						
Intervalos	**TMR**	**FR**	**RE**	**Eficiência**	**Mensagens de Rollback**	**TD (ms)**
1	2,21	7,33%	0,47	49,73%	2230	1029
2	2,19	7,11%	0,47	49,18%	2369	2903
3	2,30	7,19%	0,50	46,67%	2459	4000
4	2,24	7,26%	0,48	48,10%	2323	1256
5	2,3	7,06%	0,46	48,64%	2268	1052
6	2,28	7,20%	0,48	47,75%	2350	2000
7	2,25	7,29%	0,47	48,45%	2235	362
8	2,19	7,37%	0,47	49,10%	2206	522
9	2,22	7,13%	0,49	48,11%	2360	2477
10	2,23	6,91%	0,47	48,88%	2252	2096
Média	2,24	7,19%	0,48	48,46%	2305	1770
DP	0,04	0,14%	0,01	0,86%	80	1144

Tabela 5.1: Dados de 1 (uma) replicação do *Time Warp* com M1

dos dados de todas as replicações, estão no Apêndice B. Cada uma das tabelas é gerada calculando-se a média dos valores de 10 tabelas do tipo da tabela 5.1.

Para que as linhas de execução híbridas sejam geradas, é necessário que se compare os valores das métricas das variantes para cada protocolo, de cada tabela do Apêndice B. Em termos de implementação, pode-se considerar cada linha híbrida como um *array booleano* de n posições (n sendo igual ao número de intervalos). Cada valor deste *array*, designará qual protocolo deve ser executado a partir da comparação das métricas. Como existem 5 métricas, o protocolo que for vencedor de 3 (a maioria) das métrica terá a predileção na formação da linha de execução híbrida correspondente.

Basicamente, esta é a metodologia que encerra a primeira etapa do método definido no capítulo 3 em que uma linha hídrida é definida para cada protocolo. Após as linhas de execução híbrida de para cada modelo ser gerada, a etapa 2 do método se inicia por executar cada um dos protocolos nos intervalos definidos pela linha de execução híbrida gerada na etapa 1 do método. Os resultados da etapa 2 consistem nos custos de troca de protocolos, nos intervalos onde são determinados à ocorrer.

O roteiro de testes que será explanado nas seções 5.4.2 e 5.4.3 baseia-se, respectivamente, na primeira etapa do método, na qual as linhas híbridas são geradas, e

na segunda etapa do método, na qual as linhas híbridas são testadas.

A seção seguinte mostrará os resultados da primeira etapa, acompanhada da sua respectiva validação, e algumas análises adicionais.

5.4.2 Primeira Etapa

A primeira etapa do método proposto, consiste em executar a simulação distribuída para todos os modelos, com ambos protocolos. A partir da comparação dos seus resultados de desempenho por intervalo, é obtida uma linha de execução que indica a melhor opção de qual protocolo executar para cada intervalo de tempo para o modelo simulado. Essa linha de execução pode sugerir tanto o *Time Warp*, quanto o *Rollback* Solidário, para todos os intervalos da simulação daquele modelo, ou um arranjo híbrido que especifique qual protocolo executar em cada intervalo de simulação, de modo que haja trocas de protocolo durante a execução da simulação. Para que essa comparação seja possível, é necessário que, inicialmente, a simulação seja executada com cada um dos protocolos isoladamente de modo a gerar uma base de dados para que possam ser comparados. Os resultados da execução isolada dos modelos com cada um dos protocolos estão dispostos nas tabelas do Apêndice B deste trabalho.

Aplicando a metodologia da comparação de métricas especificada no capítulo 3 aos resultados dos experimentos isolados, obtém-se as linhas de execução com as melhores possibilidades de execução para cada intervalo. Embora a metodologia especifique que deva ser calculada uma linha de execução baseada na média para um total de n replicações, existe a possibilidade dos resultados variarem ao longo dessas replicações, de modo que possam variar de acordo com as linhas de execução escolhidas para se fazer uma comparação isolada, ou seja, considerando apenas o resultado de um replicação com o *Time Warp* e outra com o *Rollback* Solidário. Como foi realizada, para cada protocolo, 10 replicações em cada modelo, as comparações possíveis resultam em um total de 100 comparações, nas quais cada resultado de uma replicação feita para o *Time Warp* é comparada com o resultado de uma replicação do *Rollback* Solidário. Esse algoritmo pode ser compreendido com uma matriz de ordem n na qual cada elemento consiste no resultado de uma

dessas comparações. Ao final, os elementos dessa matriz são comparados, sendo anotados apenas aqueles que não foram repetidos com sua respectiva taxa de incidência (por exemplo, se uma mesma linha de execução é repetida 10 vezes dentro da matriz, então sua taxa de incidência é de 10%).

Portanto, é verificado se a linha média corresponde à linha de maior incidência nas comparações caso a caso de todas as replicações, e caso não corresponda, qual sua taxa de convergência entre as mesmas. A tabela 5.2 elenca as respectivas linhas de execução média (LM), e de maior incidência (LMI) para cada modelo, após 10 replicações. É avaliado, também, qual a convergência de uma linha em relação à outra (C) e a respectiva taxa de incidência (T.I) da linha de maior incidência. Ambos os parâmetros, C e T.I, são mostrados na mesma coluna em pares, pois cada um designa um atributo de uma das linhas. Também é mostrado, nesta tabela, o número de linhas de execução não replicadas (N), sugeridas por esse algoritmo de comparação caso a caso.

	Intervalos										**C**	
Modelos	1	2	3	4	5	6	7	8	9	10	**T.I**	**N**
M1 (LM)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
M1 (LMI)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	31%	16
M2 (LM)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
M2 (LMI)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
M3 (LM)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
M3 (LMI)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G1 (LM)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G1 (LMI)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G2 (LM)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G2 (LMI)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G3 (LM)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G3 (LMI)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
S1 (LM)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
S1 (LMI)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
S2 (LM)	TW	TW	TW	TW	RS	RS	RS	RS	RS	RS	100%	1
S2 (LMI)	TW	TW	TW	TW	RS	RS	RS	RS	RS	RS	13%	49
S3 (LM)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
S3 (LMI)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	85%	2

Tabela 5.2: Linhas de execução por cálculo de métricas

Na tabela 5.2 é evidente que, para todos os modelos, as linhas de execução médias (LM) coincidem totalmente com as linhas de execução de maior incidência (LMI) das comparações entre os dados das replicações. Os dados mostram uma predominância, em todos os intervalos, do protocolo *Time Warp* (TW) para os modelos menores **M1** e **M2**, e do protocolo *Rollback* Solidário (RS) para os modelos maiores **M3**, **G1**, **G2** e **G3**, em conformidade do que foi apontado em Azevedo (2012). Entretanto, essa tendência não se mantém somente considerando o tamanho do modelo, pois os modelos maiores de 100 processos, **S1**, **S2** e **S3** divergiram quanto a indicação de protocolos por intervalo: ao passo que o modelo **S1** indica uma predominância de desempenho do *Time Warp*, para todos os intervalos; **S2** indica linhas de execução híbridas, predominando o *Time Warp* nos primeiros quatro intervalos, e o protocolo *Rollback* Solidário nos últimos intervalos.

Analogamente à tabela 5.2, a tabela 5.3 mostra as linhas de execução médias (LM) e as linhas de execução de maior incidência (LMI), de todos os modelos testados, utilizando o critério de comparação dos tempos desperdiçados em cada intervalo, para fins de validação. Conforme explicado no capítulo 3, o tempo desperdiçado total de cada intervalo consiste na soma do tempo gasto com *rollbacks* e o tempo dispendido na transição de intervalos, quando do cálculo do GVT.

A maior diferença entre os resultados da tabela 5.2 e a tabela 5.3 é que, na tabela 5.3, a LM e a LMI dos modelos **S2** e **S3** não convergem completamente, apesar de uma alta convergência de 80%. Isso se deve, principalmente, à uma baixa taxa de incidência (11%) da LMI dos respectivos modelos, o que confere às outras 22 linhas de execução, sugeridas pelo algoritmo, um maior peso na determinação da linha de execução média.

Conforme explicado na metodologia do capítulo 3, as linhas de execução geradas por tempo desperdiçado, como na tabela 5.3, servem como forma de validar os resultados da comparação de métricas por intervalo (tabela 5.1). Essa validação implica comparar as linhas médias obtidas por cada modelo, com ambos os critérios, e analisar a sua convergência (C). A partir dessa análise é possível, não apenas verificar se as métricas refletem adequadamente um melhor desempenho dos protocolos, como averiguar quais métricas mais influem, e o quantos estas influem, na melhoria do desempenho da simulação.

	Intervalos										C	N
Modelos	1	2	3	4	5	6	7	8	9	10	**T.I**	
M1 (LM)	TW	RS	RS	RS	TW	TW	TW	TW	RS	TW	100%	1
M1 (LMI)	TW	RS	RS	RS	TW	TW	TW	TW	RS	TW	29%	15
M2 (LM)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
M2 (LMI)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
M3 (LM)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
M3 (LMI)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
G1 (LM)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G1 (LMI)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G2 (LM)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G2 (LMI)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G3 (LM)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
G3 (LMI)	RS	RS	RS	RS	RS	RS	RS	RS	RS	RS	100%	1
S1 (LM)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
S1 (LMI)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
S2 (LM)	TW	TW	RS	RS	RS	RS	RS	RS	RS	RS	80%	1
S2 (LMI)	TW	TW	TW	RS	TW	RS	RS	RS	RS	RS	11%	22
S3 (LM)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	100%	1
S3 (LMI)	TW	TW	TW	TW	TW	TW	TW	TW	TW	TW	99%	2

Tabela 5.3: Linhas de execução por tempo desperdiçado

A tabela 5.4 mostra a porcentagem de convergência entre as linhas geradas nas tabelas 5.2 e 5.3 e quais métricas prevaleceram na comparação dos intervalos para cada modelo. Os percentuais indicados entre parênteses ao lado da sigla de cada protocolo, indicam a incidência da prevalência do protocolo em questão, no total de comparações incluindo todos os intervalos de todas as replicações.

Modelos	**TMR**	**FR**	**RE**	**Eficiência**	**Mens. R**	**C**
M1	TW (100%)	RS (62,4%)	TW (79,5%)	TW (100%)	RS (99%)	60%
M2	TW (94%)	TW (92,2%)	TW (90,6%)	TW (100%)	TW (100%)	100%
M3	TW (100%)	RS (100%)	RS (100%)	RS (100%)	TW (59,2%)	0%
G1	TW (100%)	RS (100%)	RS (100%)	RS (100%)	RS (100%)	100%
G2	TW (100%)	RS (100%)	RS (100%)	TW (100%)	RS (100%)	100%
G3	TW (100%)	RS (100%)	RS (100%)	RS (100%)	RS (100%)	100%
S1	TW (100%)	RS (100%)	TW (100%)	TW (100%)	RS (100%)	100%
S2	TW (100%)	RS (100%)	TW (55,4%)	TW (100%)	RS (100%)	80%
S3	TW (100%)	RS (100%)	TW (98,5%)	TW (100%)	TW (53,7%)	100%

Tabela 5.4: Análise de convergência das linhas de execução

Os modelos para os quais a convergência das linhas foi total, ou seja de 100%, são aqueles que se enquadram nas seguintes situações:

1. Um mesmo protocolo, *Time Warp*, prevaleceu para todas as métricas, ou ao menos quatro delas, mesmo que não totalmente. Esse foi o caso dos modelos **M2** e **S3**;

2. Um protocolo, *Rollback* Solidário (modelos **G1**, **G2** e **G3**) ou *Time Warp* (modelo **S1**) prevaleceu para a maioria das métricas (na proporção de 3 para 2, ou de 4 para 1) e para cada métrica houve a prevalência total (100%) do protocolo em questão.

Nos modelos em que a convergência das linhas não foi total, pode-se notar que além da prevalência dos protocolos para as métricas ter sido na proporção de 3 para 2, nem todas as métricas tiveram incidência de 100% para o protocolo "vencedor". No caso do modelo **M1**, as métricas "frequência de *rollbacks*" (FR), "*rollbacks* por eventos" (RE) e "mensagens de *rollback*" não tiveram incidência total para o protocolo prevalente que designam. Isso significa que, em algumas comparações "caso a caso" das linhas de execução, alguns intervalos, em algumas comparações,

apresentaram resultados melhores com o seu protocolo concorrente. Este fato explica o porquê, apesar das métricas indicarem uma linha de execução média com a prevalência do *Time Warp* para todos os intervalos, a linha de validação deste modelo divergir consideravelmente da linha obtida pelas métricas.

O modelo **M3** é um modelo no qual não houve convergência entre a linha obtida pelas métricas e a linha de validação obtida pelo somatório do tempo desperdiçado em cada intervalo. Se for analisado a incidência da métrica "mensagens de *rollback*" pode-se constatar que a sua incidência foi próxima da metade dos 50% do total das comparações efetuadas. O número de mensagens de *rollback*, para ambos os protocolos ao longo dos intervalos, foi bastante próximo. Essa diferença pequena em número de mensagens, resulta em tempos de transições de intervalo bastante próximos, o que torna o tempo gasto com *rollback*, o fiel da balança no cálculo de tempo desperdiçado total. Como o "tamanho médio do *rollback*" (TMR) foi maior no caso do *Rollback* Solidário, embora FR, RE e Eficiência apresentem valores melhores que o *Time Warp*, isso indica que os *rollbacks* mais compridos do protocolo de Moreira (2005) gastaram mais tempo que os *rollbacks* mais numerosos, porém mais curtos, do *Time Warp*. Esse é um indício de que a métrica TMR tem mais impacto no desempenho do protocolo, que as métricas que são diretamente proporcionais à quantidade de *rollbacks* locais desencadeados.

Finalmente, o modelo **S2** é o mais significativo, visto que apenas esse modelo obteve linhas de execução híbridas quando da comparação dos protocolos baseada nas métricas estipuladas. A métrica RE, para esses dois modelos, mostrou incidências próximas a 50%, ao passo que as demais métricas estão bem definidas a respeito de qual protocolo teve melhor desempenho. Nesse modelo, o fato do *Time Warp* preponderar com a métrica TMR e o *Rollback* Solidário preponderar com a métrica "mensagens de *rollback*", criam uma situação de equilíbrio, que só é desempatada pela predominância de um dos dois protocolos na métrica RE, que varia razoavelmente às vezes para o *Time Warp*, às vezes para o lado do *Rollback* Solidário, embora predomine levemente o protocolo *Time Warp* para essa métrica, nesses modelos.

A análise desses modelos permite tecer conclusões a respeito do peso das métricas utilizadas no tocante ao desempenho medido pelo tempo desperdiçado total

desses experimentos. As métricas TMR e "mensagens de *rollback*" são mais significativas na influência do cálculo dos tempos desperdiçados à cada intervalo, ao passo que as métricas FR, RE e Eficiência são menos significativas na influência deste cálculo.

5.4.3 Segunda Etapa

A segunda etapa desses experimentos consistiu em executar as linhas híbridas determinadas pela primeira etapa do método e calcular os custos envolvidos na troca de protocolo. Considerando todos os modelos simulados, a quantidade de linhas de execução geradas por esse mecanismo totalizam 136 possibilidades, considerando todas as LM, LMI e as demais linhas com incidência menor. Como, porém, o modelo **S2** foi o único que gerou linhas híbridas, tanto para a comparação realizada por meio das métricas, quanto para o procedimento de validação por meio do tempo desperdiçado, as linhas deste modelo foram usadas na segunda etapa desse método.

A equação 3.8 define a centralidade da segunda etapa desse método. Com o cálculo de custos em termos de tempo, é possível comparar os tempos obtidos no termo da equação e definir se será vantajoso ou não efetuar a troca de protocolos. A tabela 5.5 mostra os custos da troca de protocolos em termos de milisegundos nos intervalos aonde essas transições são realizadas. A parte da tabela que se encontra zerada significa que não houve troca de protocolos ao final destes intervalos. Como a linha média definida pelas métricas para **S2** efetua troca de protocolos na passagem do 4^o para o 5^o intervalo, e a linha de validação do 2^o para o 3^o, somente ao final desses intervalos haverá um custo de troca de protocolos.

Linhas	**Intervalos**									
	1	2	3	4	5	6	7	8	9	10
Métricas (ms)	0	0	0	84	0	0	0	0	0	0
Validação (ms)	0	230	0	0	0	0	0	0	0	0

Tabela 5.5: Custo da troca de protocolos em milisegundos

A tabela 5.6 mostra os tempos desperdiçados totais (também em milisegundos)

ao longo dos intervalos de simulação do modelo **S2** para ambos os protocolos. Conforme explicado no capítulo 3, este tempo equivale a soma do tempo gasto com *rollback* e o tempo gasto na transição de intervalos para o cálculo do GVT.

Protocolos	**Intervalos**									
	1	2	3	4	5	6	7	8	9	10
Time Warp	360	691	2552	2862	2795	2693	6823	11917	6902	9002
Rollback Solidário	1060	1001	905	866	726	739	794	10641	1523	708

Tabela 5.6: Tempos desperdiçados por intervalo

Com base nessas duas tabelas, e utilizando a equação 3.8 para a decisão sobre a vantagem, ou não, de se efetuar a troca, é possível concluir que a troca de protocolos é vantajosa na transição do 2º para o 3º intervalo, mas não do 4º para o 5º. Caso o custo de troca excedesse a opção de se executar com o outro protocolo, a análise deveria ser refeita e a simulação re-executada de modo a considerar a troca de protocolos ao final do intervalo seguinte, anulando-a se estiver definida uma outra troca de protocolo, ao final do mesmo intervalo, no sentido inverso.

No caso do modelo **S2**, a linha definida pela análise do tempo desperdiçado, como forma de validação à linha definida por meio das métricas, representa o melhor resultado em termos de desempenho. Isto pode ser verificado ao se somar os tempos desperdiçados de todos os intervalos, incluindo os custos de troca de protocolos calculados. A tabela 5.6 mostra o tempo desperdiçado de cada linha de execução. A linha híbrida, obtida por meio dos tempos desperdiçados na primeira etapa, ganha em termos de desempenho da linha híbrida obtida por meio da análise das métricas, e das linhas que só consideram aplicar um único protocolo em sua execução.

Linha de Execução	**Tempo Desperdiçado Total**
Time Warp	47s (46596 ms)
Rollback Solidário	19s (18963 ms)
Híbrida (Métricas)	22s (21680 ms)
Híbrida (Validação)	18s (18183 ms)

Tabela 5.7: Tempos desperdiçados por linha de execução

A primeira etapa do método propôs linhas de execução que em sua maioria

convergem em sua totalidade com as linhas de validação utilizadas, embora em alguns casos a convergência não tenha sido total (modelos **M1** e **S2**), ou divergiram completamente (modelo **M3**). Esses casos à parte, levam à reflexão sobre a necessidade de se adequar o peso das métricas definidas no capítulo 3, afim de aumentar a convergência entre as linhas obtidas na primeira etapa, e consequentemente aumentar a probabilidade de se obter o melhor resultado na segunda etapa.

5.5 Considerações Finais

Neste capítulo, a metodologia desenvolvida no capítulo 3 foi usada para realizar experimentos com os modelos **M1**, **M2**, **M3**, **G1**, **G2**, **G3**, **S1**, **S2** e **S3**. Os experimentos mostraram que, na maioria dos casos, as linhas de execução determinadas pela comparação das métricas convergiam com as linhas de validação obtidas através da comparação dos respectivos tempos desperdiçados por intervalo. O caso do modelo testado na etapa 2 apresentou uma situação em que todas as trocas de protocolo se mostraram vantajosas para a linha híbrida de validação, mas não para a linha de execução definida pela comparação das métricas.

Esses resultados, e o fato de nem todos os modelos convergirem totalmente com suas linhas de validação, mostram que existe a necessidade de um refinamento metodológico que estabeleça um detalhamento maior de como e quanto cada métrica influi no acréscimo, ou decréscimo, do tempo total desperdiçado ao longo da simulação. Deste modo, será possível estabelecer pesos e critérios distintos, que correlacionem de modo mais eficiente o resultado apontado na análise de métricas com o consequente cálculo de tempo desperdiçado durante os intervalos da simulação.

6 Conclusão

Neste trabalho foi apresentado a concepção, elaboração metodológica e implementação de um método para comparação e análise dos resultados de desempenho de uma simulação distribuída executada com o *Time Warp* e com o *Rollback* Solidário. O método proposto está dividido em duas etapas, consistindo a primeira delas em se obter linhas de execuções híbridas a serem testadas na segunda etapa, considerando, então, os custos envolvidos à cada troca de protocolos definida.

No desenvolvimento desta pesquisa, os maiores desafios residiram em grande parte em adaptar o código legado de Azevedo (2012) para as novas funcionalidades necessárias à implementação do mecanismo *Protocol Swapper*, especialmente na implementação do algoritmo de Mattern (1993) para cálculo do GVT no *Time Warp* e sua respectiva adaptação para o protocolo *Rollback* Solidário. Esta implementação foi mais complexa e trabalhosa que o próprio mecanismo de troca de protocolos implementado posteriormente.

Outra dificuldade foi a adaptação do código para economia de memória, com o objetivo de evitar que erros de *stack overflow* ocorressem quando da execução da simulação de grandes modelos. Isso incluiu um controle explícito do tamanho das estruturas de dados do programa de simulação e dos protocolos e do tamanho do *heap* do Java, que foi alcançado por meio da manipulação de variáveis com escopo de classe, o que necessitou um esforço extra de sincronização e inicialização dessas variáveis em determinados contextos.

6.1 Contribuições deste Trabalho

As contribuições deste trabalho se dão em três âmbitos: em termos teóricos, com a metodologia apresentada no capítulo 3, em termos de implementação prática, com o *framework Protocol Swapper*, e em termos da massa de dados gerada em virtude dos experimentos realizados no capítulo 5, e suas respectivas conclusões no que diz respeito ao estudo de como as métricas se comportam para ambos os protocolos.

O legado teórico consiste em prover um método simples e funcional para a comparação de desempenho entre duas classes de protocolos otimistas distintos, formulando um arcabouço teórico consistente, que pode ser replicado para toda e qualquer variação desses protocolos. Isso se deve ao fato de que as formulações contidas na base dessa metodologia abarca, nas mesmas definições, características que equalizam os dois protocolos, bastante distintos em suas concepções, para fins de análise.

Em termos de implementação, este trabalho contribuiu com um aprimoramento do aplicativo desenvolvido em Azevedo (2012), atribuindo-lhe novas funcionalidades que possuem o intuito de fornecer uma gama maior de opções no que, diz respeito aos dados de desempenho levantados durante a simulação distribuida. Como explicado em capítulos anteriores, essas novas funcionalidades incluem algoritmos que tratam de questões quanto à divisão da simulação em subintervalos, o cálculo do GVT, a geração de relatórios detalhados ao final da simulação e um mecanismo de troca de protocolos. Além desses acréscimos de funcionalidade, esse aplicativo resolveu questões relacionadas à otimização do uso de memória, com variáveis de escopo mais abrangente no tratamento de eventos, sincronização e utilização do padrão *Singleton* em boa parte das classes do mecanismo proposto.

Embora precise de aprimoramentos, o sistema de métricas definido dá uma noção correta do desempenho da simulação, mostrando um alto grau de convergência para a maioria dos modelos testados no capítulo 5. A massa de dados, decorrente dos experimentos, trazem dados de desempenho significativos à respeito dos modelos simulados e dos protocolos utilizados para a simulação. Um aprofundamento

adequado na análise dos dados poderia trazer luz à questão do quanto cada métrica influi no desempenho real da simulação, e como, isso pode ser, de algum modo, equacionado.

Este trabalho, sobretudo, por dispor de uma gama de funcionalidades úteis, em termos de análise e otimização de uma simulação distribuída, o torna propício a se integrar com outras ferramentas que a complementem outras funcionalidades. Deste modo, poderia, com certo esforço de integração, se tornar parte integrante de um sistema distribuído que disponibilize ao usuário um modo escalável para simular modelos de acordo com sua necessidade.

6.2 Sugestões de Trabalhos Futuros

Em resumo a algumas considerações já feitas ao longo deste trabalho, como sugestões de trabalhos futuros seria oportuno:

- a expansão das possibilidades de aplicação deste método para outras variações de protocolos otimistas, especialmente variações do *Time Warp* e do *Rollback* Solidário;
- a expansão das possibilidades de simulação para além dos modelos genéricos testados no capítulo 5. Isso implica na possibilidade de se simular modelos mais complexos, que possuam lógicas de negócio e metas próprias.
- a adaptação deste método de comparação estático a um mecanismo de decisões dinâmicas, em que a cada intervalo seja possível tomar decisões sobre a mudança ou não de protocolo;
- o aprimoramento do sistema de decisão por meio do cálculo de métricas (se possível, equacionando o problema), de modo a buscar um aumento de convergência entre a linha obtida pela comparação de métricas e a linha de validação por meio do cálculo dos tempos desperdiçados;
- a extensão do arcabouço teórico definido neste trabalho, de modo a incluir

nele uma maneira de se possibilitar a abrangência dos protocolos conservativos, em sua grande variedade, e posterior implementação;

- um estudo comparativo de desempenho para cada protocolo com e sem a divisão por intervalos de simualação;

- o acoplamento do *framework* desenvolvido neste trabalho à uma interface gráfica, de forma análoga à proposta do trabalho de Barbosa (2012), no intuito de disponibilizar ao usuário um ambiente de simulação distribuído completo. Deste modo, o usuário teria a possibilidade de, ao mesmo tempo, dispor das facilidades de um *front-end* que o permita entrar com dados de entrada, e analisar dados de saída, de modelos matemáticos e dispor de um *software* de simulação distribuída, que rode em *back-end* em um sistema de redes com total transparência para o usuário.

Referências

AZEVEDO, M. E. C. V. d. *Projeto e Implementação dos Protocolos Otimistas Time Warp e Solidary Rolback para Simulação Distribuída*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2012.

BABAOGLU, O.; MARZULLO, K. Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. In: MULLENDER, S. (Ed.). *Distributed Systems*. 2. ed. New York: Addison-Wesley, 1993. p. 55–96.

BABULAK, E.; WANG, M. Discrete Event Simulation: State of the Art. In: GOTI, A. (Ed.). *Discrete Event Simulations*. [S.l.]: Sciyo, 2010. cap. 1, p. 1–9.

BAKER, M. et al. mpiJava: An object-oriented Java interface to MPI. In: *The 1st Java Workshop at the 13 th IPPS and 10th SPDP Conference*. [S.l.: s.n.], 1999.

BANKS, J. Introduction to Simulation. In: *Proceedings of the 31st conference on Winter simulation: Simulation - a bridge to the future*. [S.l.: s.n.], 1999. v. 1, n. 4, p. 7–13.

BANKS, J. et al. The Future of the Simulation Industry. In: *Proceedings of the 2003 Winter Simulation Conference*. [S.l.: s.n.], 2003. v. 2, p. 2033–2043.

BARBOSA, J. P. C. *Uma Ferramenta Paralela para Simulação de Eventos Discretos com Monitoramento Dinâmico de Processos*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2012.

BAUER, H.; SPORRER, C. Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving. In: *Proceedings of 26th Annual Simulation Symposium*. [S.l.: s.n.], 1993. p. 12–20.

BRYANT, R. E. *Simulation of Packet Communications Architecture Computer Systems*. [S.l.], 1977.

BUI, P. T.; LANG, S. D.; WORKMAN, D. A. A Conservative Synchronization Protocol for Dynamic Wargame Simulation. *2003 Spring Simulation Interoperability Workshop*, 2003.

CAI, W.; TURNER, S. J. An Algorithm for Distributed Discrete-Event Simulation – the "Carrier Null Message" Approach. In: *Proceedings of the SCS Multiconference on Distributed Simulation*. [S.l.: s.n.], 1990. v. 22, n. 1, p. 3–8.

CARVALHO, E. A. d. *Uma Análise sobre as Métricas para Escalonamento Dinâmico de Processos em Simulação Distribuída Usando Protocolos Otimistas*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2013.

CASTILHO, M. R. *O uso da simulação computacional como ferramenta de auxílio à tomada de decisão : aplicação em empresa de papelão ondulado*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2004.

CHANDY, K. M.; MISRA, J. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, SE-5, n. 5, p. 440–452, Sep 1979.

CHANDY, K. M.; MISRA, J. Asynchronous Distributed Simulation Via a Sequence of Parallel Computations. *Commun. ACM 24,4*, p. 198–206, Apr 1981.

CHEN, G.; SZYMANSKI, B. K. Lookback: A new way of exploiting parallelism and discrete event simulation. In: *The 16th Workshop on Parallel and Distributed Simulation. Proceedings*. [S.l.: s.n.], 2002. p. 153–162.

CHEN, G.; SZYMANSKI, B. K. Lookahead, rollback and lookback, searching for parallelism in discrete event simulation. In: *Summer Computer Simulation Conferenc*. [S.l.: s.n.], 2002b.

CHIU, G.-M.; YOUNG, C.-R. Efficient rollbackrecovery technique in distributed computing systems. *IEEE Transactions on Parallel and Distributed Systems*, v. 7, n. 6, p. 565–577, Jun 1996.

CHWIF, L.; MEDINA, A. C. *Modelagem e Simulação de Eventos Discretos – Teoria e Aplicações*. Quarta edição. [S.l.]: Editora Bravarte, 2014.

CHWIF, L.; PAUL, R. J.; BARRETTO, M. R. P. Discrete Event Simulation Model Reduction: A Causal Approach. *Simulation Practice and Theory*, v. 4, n. 7, p. 930–944, 2006.

CORMEN, T. H. et al. *Algoritmos – Teoria e Prática*. Tradução da terceira edição americana. [S.l.]: Editora Campus Ltda, 2012.

COSTA, L. C. *Teoria das Filas – Universidade Federal do Maranhão*. 2006.

COULOURIS, G.; KINDBERG, T.; DOLLIMORE, J. *Sistemas Distribuídos – Conceitos e Projeto*. Quinta edição. [S.l.]: Bookman, 2013.

CRUZ, L. B. d. *Projeto de um Framework para o Desenvolvimento de Aplicações de Simulação Distribuída*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2009.

DEELMAN, E. Improving lookahead in parallel discrete event simulations of large-scale applications using compiler analysis. In: . [S.l.: s.n.], 2001. p. 5–13.

DEITEL, P.; DEITEL, H. *Java – Como Programar*. Oitava edição. [S.l.]: Editora Pearson, 2010.

FIDGE, C. Logical time in distributed computing systems. *IEEE. Computer - Distributed computing systems: separate resources acting as one*, v. 24, n. 8, p. 28–33, 1991.

FLEISCHMANN, J.; WILSEY, P. A. Comparative analysis of periodic state saving techniques in time warp simulators. In: *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*. [S.l.: s.n.], 1995. v. 25, n. 1, p. 50–58.

FREEMAN, E. et al. *Head First Design Patterns*. [S.l.]: O'Reilly Media, 2004.

FUJIMOTO, R. M. *Parrallel and Distributed Simulation Systems*. [S.l.]: Wiley-Inter Science, 2000.

FUJIMOTO, R. M. Distributed simulation systems. In: *Proceedings of the 2003 Winter Simulation Conference*. [S.l.: s.n.], 2003. p. 124–134.

FUJIMOTO, R. M.; MALIK, A. W.; PARK, A. J. Parallel and distributed simulation in the cloud. *SCS Modeling and Simulation Magazine, Society for Modeling and Simulation*, Wiley-Inter Science, v. 1, n. 3, p. 1–10, 2010.

FUKUMOTO, S.; OHARA, M. Software Rejuvenation Schemes for Time Warp-Based PDES. In: *Proceedings of IEEE 21st Pacific Rim International Symposium on Dependable Computing (PRDC)*. Zhangjiajie: [s.n.], 2015.

GAFNI, A. Rollback Mechanisms for Optimistic Distributed Simulation Systems. In: *Proceedings of the SCS Multiconference Distributed Simulation*. [S.l.: s.n.], 1988. v. 19, n. 3, p. 61–67.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley, 1995.

GUERRA, E. *Componentes Reutilizáveis em Java com Reflexão e Anotações*. Primeira edição. [S.l.]: Casa do Código, 2014.

HAREL, D. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, p. 231–274, 1987.

ISKRA, K. A.; ALBADA, G. D. V.; SLOOT, P. M. A. Time Warp Cancellation Optimisations on High Latency Networks. In: *Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications*. [S.l.: s.n.], 2003. p. 128–135.

JAGTAP, D.; ABUGHAZALEH, N.; PONOMAREV, D. Optimization of Parallel Discrete Event Simulator for Multi-core Systems. In: *Proceedings of the 26th International Parallel and Distributed Processing Symposium*. [S.l.: s.n.], 2012. p. 520–531.

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

JUNQUEIRA, M. A. C. *Mecanismos para Migração de Processos na Simulação Distribuída*. Dissertação (Mestrado) — Universidade Federal de Itajubá, 2012.

KAWABATA, C. L. O. *Simulação Distribuída utilizando Protocolos Independentes e Troca Dinâmica nos Processos Lógicos*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - USP, São Carlos, 2005.

LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, v. 21, n. 7, p. 558–565, 1978.

LAPRE, J. M. et al. Time Warp State Restoration Via Delta Encoding. In: *Proceedings of the 2015 Winter Simulation Conference*. Piscataway, NJ, USA: [s.n.], 2015. (WSC '15).

LIN, Y. B. et al. Selecting the checkpoint interval in time warp simulation. In: *Proceedings of the 7th Workshop Parallel and Distributed Simulation*. [S.l.: s.n.], 1993. p. 3–10.

LIN, Z.; YAO, Y. An Asynchronous GVT Computing Algorithm in Neuron Time Warp-Multithread. In: *Proceedings of the 2015 Winter Simulation Conference*. Piscataway, NJ, USA: [s.n.], 2015. (WSC '15).

LOBATO, R. S. *Um método para avaliação de desempenho de protocolos de sincronização otimistas para simulação distribuída*. Tese (Doutorado) — Universidade de São Paulo, 2001.

LOBATO, R. S.; ULSON, R. S.; SANTANA, M. J. A revised taxonomy for time warp based distributed synchronization protocols. *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications, DSRT 2004*, p. 226–229, 2004.

MANIVANNAN, D.; SINGHAL, M. Quasi-synchronous checkpointing: Models, characterization, and classification. *IEEE Transactions in Parallel and Distributed Systems*, v. 10, n. 7, p. 703–713, 1999.

MATTERN, F. Virtual time and global states of distributed systems. In: *Proceedings of the Parallel and Distributed Algorithms*. [S.l.: s.n.], 1989. p. 215–226.

MATTERN, F. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing*, Orlando, FL, USA, v. 8, n. 4, p. 423–434, 1993.

MELLO, B. A. d. *Modelagem e Simulação de Sistemas*. 2001. Disponível em: <http://www.munif.com.br/munif/arquivos/ap-sim.pdf?id=319>.

MEYER, R. A.; BAGRODIA R, L. Improving Lookahead in Parallel Wireless Network Simulation. In: *Sixth IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. Montreal. Quebec. Canada: IEEE, 1998. p. 262–267.

MISRA, J. Distributed Discrete-Event Simulation. *ACM Computing Surveys*, p. 39–65, 1986.

MORAES, F. G.; SILVA, G. F.; REZENDE, T. A. *Introdução á Teoria das Filas – Universidade Federal do Mato Grosso*. 2011.

MOREIRA, E. M. *Rollback Solidário: Um Novo Protocolo Otimista para Simulação Distribuída*. Tese (Doutorado) — Universidade de São Paulo, São Paulo, 2005.

MOREIRA, E. M.; SANTANA, R. H. C.; SANTANA, M. J. Using consistent global checkpoints to synchronize processes in distributed simulation. In: *Proceedings of the 2005 Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications*. [S.l.: s.n.], 2005. p. 43–50.

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

MPI-FORUM, M. P. I. *MPI: Message Passing Interface Standard Version 3.1*. Tennessee, Knoxville, 2015. Disponível em: <http://www.mpi-forum.org/docs/mpi3.1/mpi31report.pdf>.

MPJ-EXPRESS JAVADOCS, M. P. J. *Javadocs MPJ-Express*. 2014. Disponível em: <http://mpj-express.org/docs/javadocs/index.html>.

MPJ-EXPRESS, M. P. J. *MPJ Express: An Implementation of MPI in Java*. [S.l.], Jul 2014. Linux/UNIX/Mac User Guide. Disponível em: <http://www.mpjexpress.org/docs/guides/linuxguide.pdf>.

MPJ-EXPRESS, M. P. J. *MPJ Express: An Implementation of MPI in Java*. [S.l.], Jul 2014b. Windows User Guide. Disponível em: <http://www.mpj-express.org/docs/guides/windowsguide.pdf>.

MURATA, T. Petri nets: Properties, analysis and applications. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1989. p. 541–580.

NANCE, R. E.; SARGENT, R. G. Perspectives on the evolution of simulation. *Electrical Engineering and Computer Science*, n. Paper 100, 2002. Disponível em: <http://surface.syr.edu/eecs/100/>.

NETZER, R. H. B.; XU, J. Necessary and Sufficient Conditions for Consistent Global Snapshots. *IEEE Transactions on Parallel and Distributed Systems*, v. 6, n. 2, p. 165–169, Feb 1995.

NICOL, D. M.; REYNOLDS, P. F. Problem-oriented protocol design. In: . [S.l.: s.n.], 1984. p. 471–474.

NUNES, L. F. et al. Cronosim: Uma Ferramenta de Simulação Distribuída de Eventos Discretos com Gerenciamento de Processos. In: *XLVII Simpósio Brasileiro de Pesquisa Operacional, 2015, Porto de Galinhas. Anais do XLVII Simpósio Brasileiro de Pesquisa Operacional*. Porto de Galinhas–PE: [s.n.], 2015. v. 1, p. 1–12.

PARK, A.; FUJIMOTO, R. M.; PERUMALLA, K. S. Conservative synchronization of large-scale network simulations. In: *Proceedings of the eighteenth workshop on Parallel and distributed simulation*. New York: [s.n.], 2004. p. 153–161. (PADS'04).

PERIN FILHO, C. *Introdução à Simulação de Sistemas*. [S.l.]: Editora da Unicamp, 1995.

PIENTA, R.; FUJIMOTO, R. M. On the Parallel Simulation of Scale-Free Networks. In: *Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation*. [S.l.: s.n.], 2013. p. 179–188.

PREISS, B. R.; MACINTYRE, I. D.; LOUCKS, W. M. On the Trade-off Between Time and Space in Optimistic Parallel Discrete-event Simulation. In: *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*. [S.l.: s.n.], 1992. p. 33–42.

QUAGLIA, F. Event history based sparse state saving in time warp. In: *Proceedings of the 12th Workshop on Parallel and Distributed Simulation*. [S.l.: s.n.], 1998. p. 72–79. (PADS 98).

QUAGLIA, F.; CORTELEESSA, V. Rollback-based Parallel Discrete Event Simulation by Using Hybrid State Saving. In: *Proceedings of the 9th European Simulation Symposium*. [S.l.: s.n.], 1997. p. 275–279.

RAJAEI, H. Local Time Warp: An implementation and performance analysis. 2007. (PADS'07).

RANDELL, B. System structure for software fault-tolerance. *IEEE Transactions on Software Engineering*, SE1, n. 2, p. 220–232, 1975.

REIHER, P. L. et al. Cancellation Strategies in Optimistic Execution Systems. In: *Proceedings of the 1990 Distributed Simulation Conference*. [S.l.: s.n.], 1990. v. 22, p. 112–121.

REYNOLDS, P. F. A Shared Resource Algorithm for Distributed Simulation. In: *Proceedings of the 9th Annual Symposium on Computer Architecture*. [S.l.: s.n.], 1982. p. 259–266.

RONNGREN, R. et al. Transparent Incremental State Saving in Time Warp Parallel Discrete Event Simulation. In: *Proceedings of 10th Workshop on Parallel and Distributed Simulation*. [S.l.: s.n.], 1996. p. 70–77. (PADS 96).

SAMADI, B. *Distributed Simulation*. Tese (Doutorado) — University of California, Los Angeles, 1985.

SCHMUCK, F. *The Use of Efficient Broadcast in Asynchronous Distributed Systems*. Tese (Doutorado) — Cornel University TR 88-928, Aug 1988.

SCHWARZ, R.; MATTERN, F. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, v. 7, n. 3, p. 149–174, Mar 1994.

SHAFI, A.; HAMID, K. *Running and Debugging MPJ Express with Eclipse*. [S.l.], Feb 2014. Disponível em: <http://www.mpj-express.org/docs/guides/RunningandDebuggingMPJExpresswithEclipse.pdf>.

SHAFI, A.; MANZOOR, J. Towards efficient shared memory communications in Mpj-Express. *IEEE International Symposium on Parallel and Distributed Processing*, May 2009.

SKOLD, S.; RONNGREN, R. Event sensitive state saving in time warp parallel discrete event simulation. In: *Proceedings of the 28th conference on Winter Simulation*. [S.l.: s.n.], 1996. p. 653–660.

STEINMAN, J. S. Incremental State Saving in SPEEDES using C++. In: *Proceedings of the 1993 Winter Simulation Conference*. [S.l.: s.n.], 1993. p. 687–696.

TANENBAUM, A. S. *Sistemas Operacionais Modernos*. Terceira edição. São Paulo: Pearson Prentice Hall, 2010.

TANENBAUM, A. S.; STENN, M. V. *Sistemas Distribuídos: Princípios e Paradigmas*. Segunda edição. São Paulo: Pearson - Prentice Hall, 2007.

TANG, S.; LEE, B.; HE, B. Speedup for Multi-Level Parallel Computing. In: *Proceeding of the 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. [S.l.: s.n.], 2012. p. 537–546.

TAYLOR, S. J. E. et al. Panel on grand challenges for modeling and simulation. In: *Proceedings of the 2012 Winter Simulation Conference*. [S.l.: s.n.], 2012. p. 1–15.

TEO, Y. M.; NG, Y. K. Spades/Java: Object-Oriented Parallel Discrete-Event Simulation. In: *Proceedings of the 35th Annual Simulation Symposium*. [S.l.: s.n.], 2002. p. 245–252.

WANG, J.; TROPPER, C. Optimizing Time Warp Simulation with Reinforcement Learning Techniques. In: *Proceedings of the 2007 Winter Simulation Conference*. [S.l.: s.n.], 2007. p. 577–587.

WANG, Y. M. Consistent Global Checkpoints that Contain a Given Set of Local Checkpoints. *IEEE Transactions on Computers*, v. 46, n. 4, p. 456–468, Apr 1997.

WEST, D.; PANESAR, K. Automatic Incremental State Saving. In: *Proceedings of 10th Workshop on Parallel and Distributed Simulation*. [S.l.: s.n.], 1996. p. 78–85. (PADS 96).

YAU, V. Automating parallel simulation using parallel time streams. *ACM Trans. Model.Comput. Simul.*, v. 9, n. 2, p. 171–201, 1999.

YILMAZ, L. et al. Panel: The Future of Research in Modeling & Simulation. In: *Proceedings of the 2014 Winter Simulation Conference*. [S.l.: s.n.], 2014. p. 2797–2811.

ZENG, Y.; CAI, W.; TURNER, S. J. Batch Based Cancellation: A Rollback Optimal Cancellation Scheme in Time Warp Simulation. In: *Proceedings of 18th Workshop on Parallel and Distributed Simulation*. [S.l.]: IEEE, 2004. p. 78–86. (PADSÓ4).

ZIMMERMAN, A.; KNOKE, M.; HOMMEL, G. Complete Event Ordering for Time Warp Simulation of Stochastic Discrete Event Systems. *4th Symposium on Design, Analysis, and Simulation of Distributed Systems*, Berlin, Germany, 2006. (DASD).

APÊNDICE A – Métodos Usados pelo MPJ-Express

Em cada umas das tabelas abaixo, cada função MPI correspondente virá acompanhada do método que a implementa no MPJ-Express (MPJ-EXPRESS JAVADOCS, 2014), mais a descrição de cada rotina/método apresentado.

A tabela A.1 mostra as rotinas de ambiente relativas a criação e finalização de processos.

Rotina MPI	**Método (MPJ-Express)**	**Descrição**
MPI_INIT	MPI.Init(String[ ] args)	Inicializa o MPI, com a abertura de um comunicador. O vetor "args" de *strings* traz atributos especificiados na inicialização do programa.
MPI_COMM_RANK	MPI.COMM_WORLD.Size( )	Retorna o *rank*, ou *id*, de um determinado processo no grupo do comunicador.
MPI_COMM_SIZE	MPI.COMM_WORLD.Size( )	Retorna a quantidade de processos no grupo do comunicador.
MPI_FINALIZE	MPI.Finalize( )	Finaliza o MPI. Encerra o comunicador.

Tabela A.1: Métodos de criação e finalização de processos

A tabela A.2 mostra os métodos para envio e recebimento de mensagens em uma comunicação síncrona. O MPI implementa para abordagem bloqueante quatro funções para o envio de mensagens, uma para o recebimento de mensagens e uma função de sondagem (*probe*) que verifica se alguma mensagem chegou ao *buffer* de entrada (MPI-FORUM, 2015).

Rotina MPI	**Método (MPJ-Express)**	**Descrição**

MPI_SSEND	MPI.COMM_WORLD.Ssend(java.lang.Object buf, int offset, int count, Datatype datatype, int dest, int tag)	Nesta função, o emissor informa ao receptor, que aguarda um sinal para o envio de uma mensagem, antes de enviá-la.
MPI_RSEND	MPI.COMM_WORLD.Rsend(java.lang.Object buf, int offset, int count, Datatype datatype, int dest, int tag)	Nesta função, a mensagem é enviada imediatamente para a rede de comunicação, de modo que seja necessário que a função de recebimento já tenha sido executada para permitir que o sinal informado pelo receptor tenha sido recebido.
MPI_BSEND	MPI.COMM_WORLD.Bsend(java.lang.Object buf, int offset, int count, Datatype datatype, int dest, int tag)	Nesta função, a mensagem é copiada do endereço de memória da aplicação para um *buffer* alocado pelo programa, permitindo a continuidade da execução do programa. Um sinal, quando recebido pelo receptor, faz com que a mensagem seja transmitida.

MPI_SEND	MPI.COMM_WORLD.Send(java.lang.Object buf, int offset, int count, Datatype datatype, int dest, int tag)	Nesta função, o comportamento é variável e dependente do tamanho da mensagem, podendo ser idêntico ao da função MPI_Ssend ou da função MPI_Rsend.
MPI_RECV	MPI.COMM_WORLD.Recv(java.lang.Object buf, int offset, int count, Datatype datatype, int dest, int tag)	Nesta função, o processo receptor interrompe sua execução e espera até que uma mensagem chegue pelos canais de comunicação. Assim que uma mensagem chega, ela é tratada e o processo pode dar continuidade à sua execução.

MPI_PROBE	MPI.COMM_WORLD.Probe(int source, int tag)	Nesta função, o processo receptor interrompe sua execução e espera até que seja sinalizado a chegada de uma mensagem pelos canais de comunicação. Assim que uma mensagem chega, a função notifica o processo sobre a chegada e o processo pode dar continuidade à sua execução.

Tabela A.2: Métodos para comunicação síncrona

A tabela A.3 mostra os métodos para envio e recebimento de mensagens em uma comunicação assíncrona. Analogamente ao caso síncrono, o MPI implementa para abordagem não-bloqueante quatro funções para o envio de mensagens, uma para o recebimento de mensagens e uma função de sondagem (*probe*) que verifica se alguma mensagem chegou ao *buffer* de entrada. As rotinas não-bloqueantes se diferem das rotinas de envio bloqueante por não esperarem o sinal do processo receptor. As funções de recebimento de mensagens e sondagem não-bloqueantes realizam a verificação se existem mensagens a serem recebidas, ao serem executadas. Entretanto, a computação não é interrompida caso não haja nenhuma mensagem a ser recebida, no instante da execução da rotina. Todas as funções-não bloqueantes recebem o prefixo do modo "MPI_Ixxx" (MPI-FORUM, 2015).

Nas seções seguintes segue-se a especificação dos parâmetros dos métodos utilizados pela implementação MPJ-Express, no que se refere à inicialização e finalização do MPI e à comunicação ponto a ponto, tanto para comunicação bloqueante (síncrona) quanto para a comunicação não-bloqueante(assíncrona). A especifica-

ção de cada um desses métodos está no javadocs do MPJ-Express (MPJ-EXPRESS JAVADOCS, 2014).

A.1 Inicialização e Finalização do MPI

Inicialização do MPI.

```
public static java.lang.String[] Init(java.lang.String[] args) throws
    MPIException

Descrição: Inicializa o MPI. Vínculo Java da operação MPI_INIT.

Argumentos:
1. args: argumentos do método main (public static void main(String[]
   args)).

Retorno:

Exceções:
   MPIException
```

Finalização do MPI.

```
public static void Finalize() throws MPIException

Descrição: Finaliza o MPI. Vínculo Java da operação MPI_FINALIZE.

Este método não possui argumentos.

Este método não tem parâmetros de saída.

Exceções:
   MPIException
```

Rank do processo.

```
public int Rank() throws MPIException
```

```
Descrição: Rank do processo no grupo do seu comunicador. Vínculo Java
    da operação MPI_COMM_RANK.

Este método não possui argumentos.

Retorno: rank do processo que o invoca o método no grupo do
    comunicador.

Exceções:
   MPIException
```

Tamanho do grupo.

```
public int Size() throws MPIException

Descrição: Tamanho do grupo do comunicador. Vínculo Java da operação
    MPI_COMM_SIZE.

Este método não possui argumentos.

Retorno: numero de processos no grupo do comunicador.

Throws:
   MPIException
```

A.2 Comunicação Ponto a Ponto

A.2.1 Comunicação Bloqueante

Operação de envio de mensagens bloqueante (Send).

```
public void Send(java.lang.Object buf, int offset, int count,
    Datatype datatype, int dest, int tag) throws MPIException

Descrição: Envio de mensagem (send) no modo bloqueante. Vínculo Java
    da operação MPI_SEND.

Argumentos:
```

```
1. buf: array a ser enviado no buffer de saída
2. offset: endereço inicial no buffer de saída
3. count: número de ítens a serem enviados
4. datatype: tipo de dados de cada ítem no buffer de saída
5. dest: rank, ou id, de destino
6. tag: etiqueta (rótulo) da mensagem

Este método não tem parâmetros de saída.

Explicações Adicionais: O argumento real associado com ``buf'' deve
    ser um array unidimensional. O valor offset está subscrito no
    array, definindo a posição do primeiro item da mensagem. Se o
    argumento ``datatype'' representa um tipo básico MPI, seu valor
    deve ser igual ao tipo do elemento de ``buf'' -- tanto um tipo
    primitivo quanto um tipo de referência (objeto). Se o argumento ``
    datatype'' representa um tipo derivado do MPI, seu tipo base deve
    ser igual ao tipo de elemento de ``buf''.

Exceções:
   MPIException
```

Operação de envio de mensagens bloqueante (Ssend).

```
public void Ssend(java.lang.Object buf, int offset, int count,
    Datatype datatype, int dest, int tag) throws MPIException

Descrição: Envio de mensagem (send) no modo síncrono. Vínculo Java da
     operação MPI_SSEND.

Argumentos:
1. buf: array a ser enviado no buffer de saída
2. offset: endereço inicial no buffer de saída
3. count: número de ítens a serem enviados
4. datatype: tipo de dados de cada ítem no buffer de saída
5. dest: rank, ou id, de destino
6. tag: etiqueta (rótulo) da mensagem

Este método não tem parâmetros de saída.

Exceções:
```

```
    MPIException
```

Operação de envio de mensagens bloqueante (Bsend).

```
public void Bsend(java.lang.Object buf, int offset, int count,
    Datatype datatype, int dest, int tag) throws MPIException

Descrição: Envio de mensagem (send) no modo buffer. Vínculo Java da
    operação MPI_BSEND. Esta operação copia a mensagem dentro do
    buffer (mpi.Buffer) especificada por MPI.Buffer_attach operation,
    e então envia usando o modo de comunicação padrão.

Argumentos:
1. buf: array a ser enviado no buffer de saída
2. offset: endereço inicial no buffer de saída
3. count: número de ítens a serem enviados
4. datatype: tipo de dados de cada ítem no buffer de saída
5. dest: rank, ou id, de destino
6. tag: etiqueta (rótulo) da mensagem

Este método não tem parâmetros de saída.

Exceções:
    MPIException
```

Operação de envio de mensagens bloqueante (Rsend).

```
public void Rsend(java.lang.Object buf, int offset, int count,
    Datatype datatype, int dest, int tag) throws MPIException

Descrição: Envio de mensagem (send) no modo ready. Vínculo Java da
    operação MPI_RSEND.

Argumentos:
1. buf: array a ser enviado no buffer de saída
2. offset: endereço inicial no buffer de saída
3. count: número de ítens a serem enviados
4. datatype: tipo de dados de cada ítem no buffer de saída
5. dest: rank, ou id, de destino
6. tag: etiqueta (rótulo) da mensagem
```

```
Este método não tem parâmetros de saída.

Exceções:
   MPIException
```

Operação de recebimento de mensagens bloqueante (Recv).

```
public Status Recv(java.lang.Object buf, int offset, int count,
   Datatype datatype, int dest, int tag) throws MPIException

Descrição: Recebimento de mensagem (receive) no modo bloqueante.
   Vínculo Java da operação MPI_RECV.

Argumentos:
1. buf: array a ser enviado no buffer de saída
2. offset: endereço inicial no buffer de saída
3. count: número de ítens a serem enviados
4. datatype: tipo de dados de cada ítem no buffer de saída
5. dest: rank, ou id, de destino
6. tag: etiqueta (rótulo) da mensagem

Retorno: objeto Status, indicando o status de recebimento da mensagem
   .

Exceções:
   MPIException
```

Operação de notificação da chegada de mensagens bloqueante (Probe).

```
public Status Probe(int source, int tag) throws MPIException

Descrição: Espera até que exista uma mensagem de chegada
   correspondente a um padrão específico. Vínculo Java da operação
   MPI_PROBE.

Argumentos:
1. source: rank da fonte
2. tag: etiqueta (rótulo) da mensagem
```

```
Retorno: objeto Status, de modo análogo à operação Recv.

Exceções:
   MPIException
```

A.2.2 Comunicação Não-Bloqueante

Operação de envio de mensagens não-bloqueante (Isend).

```
public Request Isend(java.lang.Object buf, int offset, int count,
   Datatype datatype, int dest, int tag) throws MPIException

Descrição: Envio de mensagem (send) no modo não-bloqueante. Vínculo
   Java da operação MPI_ISEND.

Argumentos:
1. buf: array a ser enviado no buffer de saída
2. offset:      endereço inicial no buffer de saída
3. count: número de ítens a serem enviados
4. datatype: tipo de dados de cada ítem no buffer de saída
5. dest: rank, ou id, de destino
6. tag: etiqueta (rótulo) da mensagem

Retorno: objeto Request.

Exceções:
   MPIException
```

Operação de envio de mensagens não-bloqueante (Issend).

```
public Request Issend(java.lang.Object buf, int offset, int count,
   Datatype datatype, int dest, int tag) throws MPIException

Descrição: Envio de mensagem (send) no modo síncrono não-bloqueante.
   Vínculo Java da operação MPI_ISSEND.

Argumentos:
1. buf: array a ser enviado no buffer de saída
2. offset:      endereço inicial no buffer de saída
```

```
3. count: número de ítens a serem enviados
4. datatype: tipo de dados de cada ítem no buffer de saída
5. dest: rank, ou id, de destino
6. tag: etiqueta (rótulo) da mensagem

Retorno: objeto Request.

Exceções:
   MPIException
```

Operação de envio de mensagens não-bloqueante (Ibsend).

```
public Request Ibsend(java.lang.Object buf, int offset, int count,
    Datatype datatype, int dest, int tag) throws MPIException

Descrição: Envio de mensagem (send) no modo buffer não-bloqueante.
    Vínculo Java da operação MPI_IBSEND.

Argumentos:
1. buf: array a ser enviado no buffer de saída
2. offset:      endereço inicial no buffer de saída
3. count: número de ítens a serem enviados
4. datatype: tipo de dados de cada ítem no buffer de saída
5. dest: rank, ou id, de destino
6. tag: etiqueta (rótulo) da mensagem

Retorno: objeto Request.

Exceções:
   MPIException
```

Operação de envio de mensagens não-bloqueante (Irsend).

```
public Request Irsend(java.lang.Object buf, int offset, int count,
    Datatype datatype, int dest, int tag) throws MPIException

Descrição: Envio de mensagem (send) no modo ready não bloqueante.
    Vínculo Java da operação MPI_IRSEND.

Argumentos:
```

```
1. buf: array a ser enviado no buffer de saída
2. offset:      endereço inicial no buffer de saída
3. count: número de ítens a serem enviados
4. datatype: tipo de dados de cada ítem no buffer de saída
5. dest: rank, ou id, de destino
6. tag: etiqueta (rótulo) da mensagem

Retorno: objeto Request.

Exceções:
   MPIException
```

Operação de recebimento de mensagens não-bloqueante (Irecv).

```
public Request Irecv(java.lang.Object buf, int offset, int count,
   Datatype datatype, int dest, int tag) throws MPIException

Descrição: Recebimento de mensagem (receive) no modo não-bloqueante.
   Vínculo Java da operação MPI_IRECV.

Argumentos:
1. buf: array a ser enviado no buffer de saída
2. offset:      endereço inicial no buffer de saída
3. count: número de ítens a serem enviados
4. datatype: tipo de dados de cada ítem no buffer de saída
5. dest: rank, ou id, de destino
6. tag: etiqueta (rótulo) da mensagem

Retorno: objeto Request.

Exceções:
   MPIException
```

Operação de notificação da chegada de mensagens não-bloqueante (Iprobe).

```
public Status Iprobe(int source, int tag) throws MPIException

Descrição: Verifica se existe alguma mensagem de chegada
   correspondente a um padrão específico. Vínculo Java da operação
   MPI_IPROBE.
```

```
Argumentos:
1. source: rank da fonte
2. tag: etiqueta (rótulo) da mensagem

Retorno: objeto Status.

Explicações adicionais: Se existe alguma mensagem disponível é
    retornado um objeto Status. Caso contrário, é retornado null.

Exceções:
   MPIException
```

Rotina MPI	Método (MPJ-Express)	Descrição
MPI_ISSEND	MPI.COMM_WORLD.Issend(java.lang.Object buf, int offset, int count, Datatype datatype, int dest, int tag)	Esta rotina diferencia-se da sua função homóloga, MPI_SSEND, por não interromper seu processamento após o envio da mensagem.
MPI_IRSEND	MPI.COMM_WORLD.Irsend(java.lang.Object buf, int offset, int count, Datatype datatype, int dest, int tag)	Esta rotina diferencia-se da função bloqueante homóloga, MPI_RSEND, por não interromper seu processamento após o envio da mensagem.
MPI_IBSEND	MPI.COMM_WORLD.Ibsend(java.lang.Object buf, int offset, int count, Datatype datatype, int dest, int tag)	Esta rotina diferencia-se da função bloqueante homóloga, MPI_BSEND, por não interromper seu processamento após o envio da mensagem.
MPI_ISEND	MPI.COMM_WORLD.Isend(java.lang.Object buf, int offset, int count, Datatype datatype, int dest, int tag)	Esta rotina diferencia-se da função bloqueante homóloga, MPI_SEND, por não interromper seu processamento após o envio da mensagem.
MPI_IRECV	MPI.COMM_WORLD.Irecv(java.lang.Object buf, int offset, int count, Datatype datatype, int dest, int tag)	Esta rotina diferencia-se da função bloqueante homóloga, MPI_RECV, por não interromper seu processamento após a verificação de mensagens a serem recebidas.
MPI_IPROBE	MPI.COMM_WORLD.Iprobe(int source, int tag)	Esta rotina diferencia-se da função bloqueante homóloga, MPI_PROBE, por não interromper seu processamento após a verificação de mensagens a serem recebidas.

Tabela A.3: Métodos para comunicação assíncrona

APÊNDICE B – Dados por Intervalo

Os dados apresentados nas tabelas a seguir são os resultados das simulações para os nove modelos definidos no capítulo 5: **M1**, **M2** e **M3**, **G1**, **G2** e **G3**, **S1**, **S2**, e **S3**. Esses resultados foram obtidos através da média de um total de 10 replicações de 10 minutos para cada modelo simulado, divididos em intervalos de 1 minuto cada. Esses resultados expressam o valor das métricas utilizadas para comparar os dois protocolos, adicionadas do tempo total desperdiçado no decorrer da simulação (expressos em milisegundos). Esses valores consistem da média das replicações e o seus respectivos desvios padrões.

Na coluna com um asterisco (*), constam enumerados os intervalos de 1 a 10, sendo os dados da tabela, uma média das 10 replicações para cada modelo. Cada uma das tabelas foi gerada por meio de outras 10, nas quais constam os dados de cada replicação. Os dados de cada replicação individual não será apresentada neste trabalho por uma questão de limitação de espaço, e por uma questão de síntese e objetividade na linha de raciocínio necessária à condução da parte experimental deste trabalho.

Modelo M1 – TIME WARP – Dados de todos os intervalos

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	2,23	0,02	7,3%	0,1%	0,46	0,01	49,2%	0,2%	2254	25	1113	297
2	2,21	0,04	7,2%	0,1%	0,47	0,01	49,1%	0,1%	2362	72	2467	510
3	2,26	0,05	7,2%	0,1%	0,51	0,01	46,5%	0,2%	2446	70	3649	1176
4	2,23	0,03	7,2%	0,1%	0,49	0,01	48,0%	0,3%	2349	57	1998	841
5	2,29	0,01	7,1%	0,1%	0,46	0,01	48,6%	0,4%	2253	52	947	621
6	2,26	0,03	7,2%	0,1%	0,48	0,01	47,9%	0,2%	2324	19	1709	869
7	2,20	0,03	7,2%	0,1%	0,48	0,01	48,6%	0,4%	2241	21	408	295
8	2,20	0,02	7,3%	0,2%	0,47	0,01	49,1%	0,2%	2238	32	968	366
9	2,21	0,02	7,2%	0,1%	0,48	0,01	48,4%	0,4%	2355	44	2452	757
10	2,25	0,03	7,1%	0,2%	0,47	0,01	48,7%	0,4%	2268	62	1491	515

* - Intervalos da Simulação

Tabela B.1: Modelo M1 - Resultados da Simulação para o *Time Warp*

Modelo M1 – SOLIDARY ROLLBACK – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	6,00	0,06	8,2%	0,1%	1,16	1,66	20,8%	0,3%	2158	18	2033	21
2	6,03	0,47	8,0%	0,4%	0,61	0,02	21,4%	1,1%	2162	71	2082	73
3	6,99	1,57	7,0%	1,0%	0,60	0,03	19,1%	3,2%	2084	65	2028	57
4	10,98	5,07	5,9%	1,8%	0,55	0,09	16,4%	3,9%	1916	293	1897	261
5	14,77	3,52	6,6%	6,9%	0,46	0,07	13,6%	2,3%	1626	278	1625	274
6	12,25	3,01	6,8%	6,3%	0,50	0,09	14,2%	2,3%	1737	320	1710	292
7	12,23	3,40	5,1%	1,1%	0,51	0,07	14,5%	2,0%	1809	199	1758	160
8	11,09	4,76	5,5%	1,4%	0,52	0,09	16,0%	3,1%	1853	252	1765	197
9	10,07	2,90	5,8%	1,1%	0,53	0,06	16,6%	3,1%	1970	143	1876	110
10	8,01	2,63	7,9%	4,4%	0,58	0,05	18,3%	3,0%	2094	424	1974	385

* - Intervalos da Simulação

Tabela B.2: Modelo M1 - Resultados da Simulação para o *Rollback* Solidário

Modelo M2 – TIME WARP – Dados de todos os intervalos

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	1,96	0,02	6,2%	1,2%	0,07	0,02	88,1%	2,2%	0	0	27	3
2	1,95	0,02	8,4%	1,8%	0,10	0,03	83,7%	3,5%	0	0	23	4
3	1,95	0,01	9,3%	2,0%	0,12	0,03	81,8%	3,8%	0	0	22	5
4	1,95	0,03	9,3%	2,9%	0,12	0,04	81,6%	5,8%	0	0	24	5
5	1,96	0,01	9,6%	3,8%	0,12	0,05	81,3%	7,3%	0	0	21	3
6	1,98	0,02	9,6%	3,2%	0,14	0,06	81,1%	6,2%	0	0	23	6
7	1,96	0,01	9,9%	3,2%	0,13	0,05	80,7%	6,2%	0	0	21	3
8	1,97	0,03	10,8%	2,6%	0,14	0,04	78,7%	5,1%	0	0	28	12
9	1,97	0,02	11,3%	2,5%	0,15	0,04	77,7%	4,9%	0	0	22	4
10	1,96	0,01	12,1%	2,7%	0,16	0,05	76,3%	5,3%	0	0	22	5

* - Intervalos da Simulação

Tabela B.3: Modelo M2 - Resultados da Simulação para o *Time Warp*

Modelo M2 – SOLIDARY ROLLBACK – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	4,42	0,71	6,6%	1,2%	0,06	0,01	78,7%	1,2%	1317	251	581	94
2	2,48	0,67	14,4%	4,7%	0,18	0,07	71,4%	4,3%	2510	738	977	294
3	2,51	0,59	16,1%	2,5%	0,23	0,04	65,1%	4,4%	3084	446	1155	159
4	2,76	0,71	16,4%	3,7%	0,23	0,05	61,6%	1,6%	3118	543	1107	181
5	2,49	0,51	17,7%	3,1%	0,25	0,04	62,1%	1,4%	3326	398	1225	151
6	2,59	0,39	17,1%	2,4%	0,25	0,04	61,5%	1,6%	3323	423	1239	159
7	2,50	0,49	17,6%	2,9%	0,25	0,04	62,0%	1,4%	3374	384	1263	150
8	2,44	0,24	18,2%	1,8%	0,26	0,02	61,8%	2,5%	3313	259	1242	77
9	2,32	0,22	18,6%	1,8%	0,26	0,02	62,8%	2,0%	3413	220	1280	75
10	2,24	0,27	19,1%	1,6%	0,27	0,02	62,5%	2,0%	3546	219	1323	76

* - Intervalos da Simulação

Tabela B.4: Modelo M2 - Resultados da Simulação para o *Rollback* Solidário

Modelo M3 – TIME WARP – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	1,06	0,01	28,9%	0,2%	1,48	0,02	39,1%	0,1%	9693	28	213	94
2	1,12	0,03	29,9%	0,4%	1,38	0,03	39,3%	0,0%	9909	15	85	10
3	1,15	0,03	30,7%	0,4%	1,34	0,07	38,4%	2,8%	9285	14	82	16
4	1,16	0,04	31,3%	0,7%	1,32	0,05	39,5%	0,1%	9462	17	112	27
5	1,19	0,04	31,5%	0,5%	1,29	0,05	39,5%	0,1%	9596	14	72	30
6	1,21	0,05	31,7%	0,6%	1,28	0,05	39,3%	0,1%	9575	47	59	6
7	1,21	0,05	32,0%	0,5%	1,28	0,06	39,4%	0,1%	9491	22	50	3
8	1,21	0,05	32,2%	0,6%	1,26	0,07	39,4%	0,1%	9770	31	60	15
9	1,21	0,06	32,2%	0,8%	1,29	0,07	39,2%	0,1%	9852	18	107	32
10	1,20	0,06	32,4%	0,9%	1,28	0,06	39,4%	0,1%	9422	39	48	8

* - Intervalos da Simulação

Tabela B.5: Modelo M3 - Resultados da Simulação para o *Time Warp*

Modelo M3 – SOLIDARY ROLLBACK – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	1,26	0,04	21,6%	1,0%	0,72	0,04	52,5%	2,1%	9158	383	2021	19
2	1,38	0,05	19,2%	0,5%	0,80	0,04	47,8%	1,8%	9759	527	2001	71
3	1,42	0,09	18,8%	0,6%	0,80	0,01	46,7%	1,5%	10096	172	2055	47
4	1,47	0,12	18,8%	0,8%	0,82	0,02	45,5%	2,0%	9727	409	1960	73
5	1,48	0,10	19,0%	0,9%	0,81	0,03	45,7%	1,7%	9653	584	1946	81
6	1,55	0,15	18,8%	1,2%	0,82	0,03	44,2%	1,8%	9587	494	1927	86
7	1,49	0,07	18,1%	0,7%	0,82	0,01	44,9%	1,1%	10010	246	1993	44
8	1,53	0,08	18,1%	0,8%	0,80	0,02	45,0%	1,2%	9787	271	1968	41
9	1,57	0,04	18,3%	1,5%	0,80	0,03	44,3%	0,5%	9697	589	1939	96
10	1,53	0,06	18,3%	1,1%	0,81	0,03	44,6%	1,5%	9847	448	1951	75

* - Intervalos da Simulação

Tabela B.6: Modelo M3 - Resultados da Simulação para o *Rollback* Solidário

Modelo G1 – TIME WARP – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	1,57	0,02	25,5%	0,4%	1,62	0,09	28,3%	1,1%	58015	2205	26603	3863
2	1,64	0,05	24,4%	1,3%	1,64	0,13	26,1%	4,1%	62328	4669	37165	14622
3	1,64	0,04	24,6%	0,9%	1,72	0,04	26,2%	0,6%	64659	2716	37849	15356
4	1,65	0,05	24,1%	1,7%	1,72	0,07	26,1%	0,8%	64343	3618	44127	22251
5	1,70	0,06	23,0%	1,3%	1,72	0,13	25,6%	1,2%	65249	4257	47865	18749
6	1,74	0,05	22,3%	0,9%	1,69	0,14	25,6%	1,4%	66034	4413	49159	18595
7	1,73	0,06	22,7%	1,2%	1,75	0,17	25,0%	1,5%	67611	5507	57887	16647
8	1,69	0,04	23,2%	0,8%	1,85	0,11	24,3%	1,0%	69401	5863	59753	14895
9	1,72	0,07	22,8%	1,5%	1,83	0,17	23,9%	1,4%	70627	4981	54219	14107
10	1,72	0,05	22,2%	1,4%	1,80	0,16	24,8%	0,9%	69697	4200	59458	13837

* - Intervalos da Simulação

Tabela B.7: Modelo G1 - Resultados da Simulação para o *Time Warp*

Modelo G1 – SOLIDARY ROLLBACK – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	4,67	0,18	6,8%	0,1%	0,29	0,01	42,5%	1,8%	10200	150	1076	23
2	5,05	0,35	7,5%	0,6%	0,28	0,05	41,7%	3,9%	8958	1281	939	141
3	5,59	0,66	8,1%	0,8%	0,24	0,05	43,2%	3,8%	7612	1607	795	191
4	5,76	0,64	8,5%	0,9%	0,23	0,06	44,3%	5,7%	7013	1784	716	193
5	5,49	0,55	8,6%	0,7%	0,22	0,05	46,3%	3,7%	6937	1413	706	140
6	6,04	0,50	9,1%	0,7%	0,18	0,05	48,4%	5,8%	5451	1486	1115	908
7	6,07	0,80	9,1%	0,7%	0,19	0,04	47,7%	3,4%	5775	1315	680	133
8	6,18	0,71	9,3%	0,6%	0,19	0,03	46,7%	2,9%	5737	999	599	109
9	6,15	0,58	8,9%	0,7%	0,21	0,04	44,6%	3,7%	6209	1172	683	117
10	6,06	0,53	9,2%	0,5%	0,19	0,03	47,5%	3,7%	5747	800	607	97

* - Intervalos da Simulação

Tabela B.8: Modelo G1 - Resultados da Simulação para o *Rollback* Solidário

Modelo G2 – TIME WARP – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	1,53	0,03	17,4%	0,3%	0,54	0,01	54,9%	0,6%	44124	1338	66708	15705
2	1,66	0,04	17,3%	0,5%	0,53	0,03	53,5%	1,6%	50622	2819	70277	19098
3	1,74	0,08	16,8%	0,7%	0,53	0,02	52,3%	0,9%	52582	2388	80999	24226
4	1,76	0,07	16,5%	0,9%	0,54	0,04	47,9%	11,2%	55014	3219	91572	24861
5	1,77	0,08	16,5%	0,8%	0,55	0,04	50,8%	1,7%	57266	3262	91748	22851
6	1,81	0,09	16,4%	0,9%	0,56	0,06	50,1%	2,0%	59125	4390	92722	22855
7	1,85	0,11	16,4%	0,9%	0,54	0,06	50,1%	2,3%	58919	5003	94782	19536
8	1,84	0,07	16,5%	1,3%	0,55	0,06	49,8%	2,3%	59165	4631	102100	14748
9	1,83	0,09	16,3%	0,5%	0,55	0,05	50,1%	2,0%	59546	3699	107908	8746
10	1,90	0,07	15,6%	0,7%	0,54	0,04	49,6%	1,7%	58396	3014	110178	3375

* - Intervalos da Simulação

Tabela B.9: Modelo G2 - Resultados da Simulação para o *Time Warp*

Modelo G2 – SOLIDARY ROLLBACK – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	7,07	0,12	5,0%	0,0%	0,33	0,02	30,2%	1,3%	10331	104	1131	29
2	8,25	0,93	5,7%	0,5%	0,27	0,05	31,2%	2,1%	8272	1363	887	153
3	9,02	1,33	6,2%	0,8%	0,22	0,06	34,5%	3,6%	6881	1837	754	208
4	8,93	1,32	6,4%	0,9%	0,21	0,06	36,2%	3,4%	6683	2032	706	213
5	9,63	1,40	6,7%	0,8%	0,20	0,06	34,9%	2,6%	6069	1837	638	198
6	9,34	0,96	6,8%	0,5%	0,18	0,04	37,5%	4,2%	6123	2013	1810	3628
7	9,45	1,30	6,8%	0,6%	0,19	0,04	36,0%	2,6%	5772	1190	671	171
8	9,41	0,97	7,0%	0,5%	0,18	0,04	38,3%	3,7%	5581	1125	679	307
9	9,63	1,17	7,0%	0,5%	0,17	0,03	35,8%	10,3%	5295	968	603	123
10	9,56	1,04	7,0%	0,5%	0,17	0,04	38,6%	4,1%	5445	1213	600	125

* - Intervalos da Simulação

Tabela B.10: Modelo G2 - Resultados da Simulação para o *Rollback* Solidário

Modelo G3 – TIME WARP – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	1,89	0,02	22,7%	0,2%	1,68	0,03	23,9%	0,3%	95335	1637	138173	10700
2	2,00	0,08	20,1%	1,6%	1,61	0,07	23,7%	0,5%	100692	3195	160478	8599
3	2,04	0,05	20,0%	1,1%	1,63	0,10	23,0%	1,4%	102394	4081	167049	22360
4	2,10	0,06	19,4%	0,8%	1,58	0,08	23,2%	0,7%	103742	3416	175704	20502
5	2,13	0,07	18,4%	1,1%	1,59	0,06	22,9%	0,7%	104685	3492	183498	13400
6	2,14	0,07	18,6%	1,4%	1,63	0,06	22,4%	0,6%	109210	3674	186578	10445
7	2,17	0,05	18,4%	0,8%	1,66	0,08	21,8%	0,7%	110306	4589	189647	15247
8	2,19	0,06	17,7%	1,5%	1,56	0,11	22,7%	0,8%	109286	3223	183974	12450
9	2,21	0,10	18,1%	1,6%	1,61	0,12	21,8%	1,2%	112417	3683	188979	6834
10	2,19	0,09	18,6%	0,9%	1,64	0,13	21,9%	1,2%	113072	6520	190082	4511

* - Intervalos da Simulação

Tabela B.11: Modelo G3 - Resultados da Simulação para o *Time Warp*

Modelo G3 – SOLIDARY ROLLBACK – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	6,33	0,17	6,8%	0,0%	0,21	0,01	42,9%	1,2%	7596	124	1123	19
2	6,61	0,29	7,2%	0,2%	0,18	0,02	45,4%	2,6%	6366	510	914	83
3	6,59	0,55	7,5%	0,1%	0,15	0,01	50,9%	3,9%	5493	372	784	66
4	6,62	0,64	7,5%	0,2%	0,15	0,01	51,7%	2,9%	5483	606	769	94
5	6,36	0,60	7,7%	0,3%	0,12	0,02	56,3%	5,2%	4985	532	728	74
6	7,11	1,33	7,5%	0,2%	0,12	0,02	54,1%	5,2%	5426	1406	1935	2557
7	6,97	0,59	7,7%	0,2%	0,13	0,02	53,5%	3,6%	4723	573	805	290
8	6,59	0,53	7,8%	0,3%	0,13	0,02	55,1%	5,0%	4891	608	698	86
9	6,64	0,79	7,9%	0,2%	0,12	0,02	55,4%	5,3%	4772	492	730	111
10	6,56	0,54	7,9%	0,3%	0,12	0,03	57,5%	5,3%	4549	659	658	107

* - Intervalos da Simulação

Tabela B.12: Modelo G3 - Resultados da Simulação para o *Rollback* Solidário

Modelo S1 – TIME WARP – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	1,30	0,02	10,8%	0,2%	0,29	0,01	72,5%	1,1%	22756	792	159	25
2	1,29	0,02	10,7%	0,1%	0,28	0,01	73,4%	0,8%	22309	694	111	35
3	1,31	0,01	10,7%	0,1%	0,29	0,01	72,4%	0,8%	23001	621	91	44
4	1,28	0,02	10,6%	0,1%	0,28	0,01	73,6%	0,9%	22333	696	108	105
5	1,30	0,02	10,8%	0,1%	0,30	0,01	72,2%	0,8%	23328	601	85	29
6	1,30	0,02	10,6%	0,2%	0,29	0,01	72,8%	1,1%	22725	859	96	30
7	1,31	0,02	10,6%	0,2%	0,29	0,02	72,4%	1,3%	22864	1085	127	37
8	1,31	0,02	10,7%	0,1%	0,30	0,01	72,1%	1,3%	23106	924	161	42
9	1,30	0,02	10,5%	0,2%	0,28	0,02	73,1%	1,2%	22458	926	152	44
10	1,31	0,02	10,7%	0,1%	0,29	0,01	72,3%	0,6%	23081	376	147	39

* - Intervalos da Simulação

Tabela B.13: Modelo S1 - Resultados da Simulação para o *Time Warp*

Modelo S1 – SOLIDARY ROLLBACK – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	6,50	0,11	3,0%	0,0%	8,31	2,54	2,0%	0,5%	18289	255	1345	50
2	7,09	0,15	3,0%	0,1%	5,85	1,84	2,6%	0,8%	17217	438	1171	45
3	7,17	0,25	3,0%	0,1%	5,52	1,85	2,6%	0,7%	17179	759	1155	67
4	6,92	0,17	3,1%	0,1%	6,64	1,75	2,3%	0,6%	17638	514	1142	42
5	6,85	0,26	3,1%	0,2%	5,23	1,46	2,8%	0,6%	17682	1023	1151	82
6	6,96	0,80	3,3%	0,4%	5,98	2,58	2,8%	1,1%	17189	2403	1118	173
7	7,51	1,43	3,6%	0,7%	5,07	2,35	3,2%	1,2%	15707	3695	1013	263
8	7,27	1,26	3,6%	0,5%	5,50	2,27	2,8%	0,9%	16514	2670	2736	5196
9	9,68	2,98	3,4%	0,5%	6,28	4,65	3,2%	1,9%	11496	5240	2042	1052
10	7,17	1,34	3,6%	0,6%	5,20	2,11	3,4%	2,5%	15297	2811	1332	523

* - Intervalos da Simulação

Tabela B.14: Modelo S1 - Resultados da Simulação para o *Rollback* Solidário

Modelo S2 – TIME WARP – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	1,54	0,02	15,3%	0,6%	0,74	0,08	46,7%	2,4%	42414	4385	360	344
2	1,58	0,02	17,1%	0,5%	1,02	0,12	38,6%	2,7%	48350	2731	691	835
3	1,55	0,02	17,7%	1,2%	1,17	0,25	37,4%	4,2%	50337	5809	2552	6842
4	1,55	0,04	18,4%	1,4%	1,20	0,20	35,6%	3,5%	51962	5370	2862	6041
5	1,52	0,11	18,7%	1,2%	1,21	0,20	35,8%	3,1%	52459	4267	2795	4986
6	1,53	0,04	19,0%	1,4%	1,22	0,16	35,3%	2,5%	55277	5550	2693	3953
7	1,54	0,04	19,6%	1,3%	1,37	0,22	32,7%	3,3%	59159	6160	6823	7840
8	1,55	0,04	19,9%	1,0%	1,45	0,20	31,2%	2,7%	61762	5617	11917	9232
9	1,55	0,04	20,1%	0,9%	1,44	0,14	30,3%	2,9%	62161	5601	6902	7687
10	1,53	0,04	20,5%	1,4%	1,62	0,10	29,2%	4,3%	63886	6641	9002	10729

* - Intervalos da Simulação

Tabela B.15: Modelo S2 - Resultados da Simulação para o *Time Warp*

Modelo S2 – SOLIDARY ROLLBACK – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	7,05	0,20	3,3%	0,1%	1,52	0,19	8,6%	1,3%	15252	227	1060	54
2	7,46	0,15	3,4%	0,1%	1,40	0,19	8,6%	1,0%	14489	410	1001	103
3	7,60	0,59	3,5%	0,2%	1,39	0,23	8,9%	1,3%	14084	1313	905	98
4	7,91	1,07	3,7%	0,5%	1,37	0,23	8,6%	0,7%	13586	2304	866	155
5	8,65	1,59	4,2%	0,9%	1,13	0,41	10,3%	2,7%	11969	3228	726	220
6	8,42	1,30	4,1%	0,8%	1,14	0,33	10,1%	2,4%	12018	2759	739	186
7	8,15	0,78	3,9%	0,4%	1,32	0,27	8,7%	1,1%	12710	1974	794	134
8	10,20	1,88	4,1%	0,5%	1,29	0,62	8,8%	3,8%	16489	11869	10641	10197
9	8,45	1,50	4,2%	0,9%	1,09	0,27	10,3%	1,7%	11179	2684	1523	883
10	9,10	1,77	4,7%	1,1%	1,16	0,42	9,5%	2,0%	10904	3107	708	200

* - Intervalos da Simulação

Tabela B.16: Modelo S2 - Resultados da Simulação para o *Rollback* Solidário

Modelo S3 – TIME WARP – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	1,55	0,02	15,6%	0,9%	0,79	0,16	45,3%	4,5%	42727	4342	360	344
2	1,58	0,02	17,2%	0,5%	1,03	0,11	38,5%	2,6%	48350	2731	691	835
3	1,55	0,02	17,8%	1,3%	1,12	0,22	37,2%	4,5%	50719	6038	2552	6842
4	1,55	0,04	18,4%	1,4%	1,20	0,20	35,6%	3,5%	52230	5007	2862	6041
5	1,55	0,04	18,7%	1,2%	1,18	0,18	35,8%	3,1%	52459	4267	2795	4986
6	1,53	0,04	19,0%	1,4%	1,21	0,16	35,5%	2,5%	55277	5550	2693	3953
7	1,55	0,04	19,6%	1,3%	1,35	0,23	32,7%	3,3%	59159	6160	6823	7840
8	1,52	0,08	20,0%	1,0%	1,49	0,17	30,2%	4,4%	61762	5617	11917	9232
9	1,55	0,04	20,1%	0,9%	1,45	0,15	31,3%	2,2%	62615	5800	6902	7687
10	1,53	0,03	20,5%	1,4%	1,52	0,29	30,0%	3,2%	63859	6610	9002	10729

* - Intervalos da Simulação

Tabela B.17: Modelo S3 - Resultados da Simulação para o *Time Warp*

Modelo S3 – SOLIDARY ROLLBACK – Intervalos TOTAL

	TMR		FR		RE		Eficiência		Mens. R		TD (ms)	
*	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P	Média	D.P
1	7,05	0,20	3,3%	0,1%	1,52	0,19	8,7%	1,3%	15252	227	1060	54
2	7,46	0,15	3,4%	0,1%	1,40	0,19	8,9%	1,0%	14489	410	1001	103
3	7,60	0,59	3,5%	0,2%	1,39	0,23	8,9%	1,3%	14084	1313	905	98
4	7,91	1,07	3,7%	0,5%	1,37	0,23	8,6%	0,7%	13586	2304	866	155
5	8,65	1,59	4,2%	0,9%	1,13	0,41	10,3%	2,7%	11892	3248	726	220
6	8,58	1,29	4,1%	0,7%	1,16	0,34	9,9%	2,5%	11936	2680	739	186
7	8,15	0,78	3,9%	0,4%	1,32	0,27	8,3%	1,9%	12710	1974	794	134
8	10,20	1,88	4,2%	0,5%	1,29	0,62	8,7%	3,8%	16489	11869	10641	10197
9	8,45	1,50	3,6%	1,3%	1,09	0,27	10,3%	1,7%	11179	2684	1523	883
10	9,10	1,77	4,4%	0,8%	1,16	0,42	9,5%	2,0%	10904	3107	708	200

* - Intervalos da Simulação

Tabela B.18: Modelo S3 - Resultados da Simulação para o *Rollback* Solidário