UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GUILHERME BIZZANI

IDENTIFICAÇÃO DE ESTADOS SEGUROS PARA REDUZIR A CRIAÇÃO DE CHECKPOINTS SEM VALOR.

CHAPECÓ
2015

GUILHERME BIZZANI

IDENTIFICAÇÃO DE ESTADOS SEGUROS PARA REDUZIR A CRIAÇÃO DE CHECKPOINTS SEM VALOR.

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Braulio Adriano de Mello

CHAPECÓ
2015

GUILHERME BIZZANI

IDENTIFICAÇÃO DE ESTADOS SEGUROS PARA REDUZIR A CRIAÇÃO DE CHECKPOINTS SEM VALOR.

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Braulio Adriano de Mello

Este trabalho de conclusão de curso foi defendido e aprovado pela banca em: ____/____/____

BANCA EXAMINADORA:

_____
Dr. Braulio Adriano de Mello - UFFS

_____
Dr. Claunir Pavan - UFFS

_____
Ma. Emilio Wuerges - UFFS

RESUMO

RESUMOOOO

Palavras-chave: Checkpoints. Sistemas distribuidos. Simulação.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Tema

Este trabalho aborda estratégias de identificação de estados seguros para a criação de Checkpoints em sistemas de simulação computacional distribuído, com o intuito de evitar a criação de Checkpoints sem valor.

1.2 Contextualização

Simulações computacionais tem o propósito de permitir a representação do comportamento de um sistema com o uso de modelos de representação. Um modelo incorpora características do sistema que representa, possuindo então a capacidade de "imitar" seu comportamento.

Tendo cada vez mais integração entre áreas de conhecimento distintas, existe um maior incentivo para o uso de modelos heterogêneos de simulação. Modelos heterogêneos permitem que os elementos da simulação diferenciem quanto sua linguagem de programação, sua interface e na maneira em que trocam mensagens, podendo assim integrar áreas de conhecimento distintas em uma mesma aplicação [Reynolds Jr, 1988].

Com o objetivo de reduzir o tempo da simulação, o modelo da simulação pode ser dividido em elementos distintos que podem ser executados em sistemas distribuídos [Fujimoto, 1998].

Durante uma simulação computacional distribuída, podem ocorrer falhas de violação de tempo na troca de mensagens entre os processos. Para evitar que seja necessário reiniciar totalmente a computação, são utilizados métodos que permitem utilizar processos de Rollback. Os métodos mais utilizados são por meio da criação de Logs e criação de Checkpoint. Estes métodos consistem em criar pontos de restauração durante a computação, os quais podem ser restaurados caso necessário, evitando assim a necessidade de reiniciar totalmente a simulação.

Segundo [Elnozahy et al., 2002], existem três abordagens principais para a criação de checkpoints, são estes os checkpoints Coordenados, que orquestram a criação de checkpoints entre todos os elementos, podendo causar overhead de processamento. Checkpoints Não-coordenados, que garantem a autonomia para que cada elemento crie seus checkpoints quando achar conveniente, mas checkpoints não coordenados possuem desvantagens, uma delas é o efeito dominó, que pode levar ao reinicio da computação, outra desvantagem é a criação de

checkpoints inúteis e por fim o problema que pode ser necessário um coletor de lixo para apagar os checkpoints que não serão mais úteis. E por fim, Checkpoints induzidos a comunicação, que procuram utilizar as mensagens da aplicação para criar seus checkpoints.

A proposta do trabalho foi desenvolvida sobre a plataforma do DCB, Distributed Co-simulation Backbone, que se trata de uma arquitetura de co-simulação heterogênea [Mello, 2005]. O DCB foi inspirado na High Level Architeture (HLA). A HLA é um padrão IEEE para arquitetura de sistemas de simulação distribuídos [Dahmann et al., 1997]. O DCB é distribuído, uma vez que permite que os modelos estejam distribuídos logicamente e fisicamente, é heterogêneo, permitindo que seus elementos sejam desenvolvidos em linguagem de programação diferentes, possuam interfaces distintas ou até mesmo interajam com elementos reais de simulação. Seus elementos assíncronos seguem a política de checkpoints quanto a recuperação pós falha.

Atualmente o DCB cria seus checkpoints de forma não coordenada, levando em consideração apenas o tempo de simulação e a quantia de troca de mensagens entre os elementos. Isso pode levar ao efeito dominó e à criação de checkpoints inúteis, uma vez que não é garantida a criação de checkpoints consistentes.

1.3 Objetivos

1.3.1 Geral

Desenvolver um mecanismo de identificação de estados seguros para a criação de checkpoints no DCB reduzindo a probabilidade de desperdício de processamento com a geração de checkpoints inúteis e garantindo o funcionamento da operação de Rollback.

1.3.2 Específicos

- Estudar técnicas de identificação de estados seguros.
- Identificar estratégias de criação de Checkpoints a partir de estados seguros.
- Selecionar uma estratégia alinhada com as características do DCB.
- Implementar a criação de Checkpoints com base em estados seguros no DCB.
- Validar a solução por meio de estudos de caso.

1.4 Justificativa

A identificação de estados seguros através do uso de Checkpoints Coordenados pode ser fácil, porém o overhead de processamento e envio de mensagens necessários para que seja criado estes checkpoints é muito alto. Neste sentido, os estudos de métodos de criação de checkpoints não-coordenados podem ser de grande valor, não apenas para simulações, mas para sistemas distribuídos em geral, uma vez que não tenha desperdício de processamento e garanta que o sistema seja tolerante a falhas.

1.5 Estrutura do Trabalho

Este trabalho está dividido em 5 capítulos. No Capítulo 1 é apresentado o tema, a contextualização ao problema seguido dos objetivos e justificativa. O Capítulo 2 apresenta o referencial teórico. No Capítulo 3 é apresentado os trabalhos relacionados. No Capítulo 4 é apresentada a metodologia da pesquisa, as etapas de desenvolvimento do projeto, cronograma de trabalho e orçamento do projeto. No Capítulo 5 são apresentadas as referências bibliográficas.

2 REFERENCIAL TEÓRICO

2.1 Simulação

Simulação consiste em empregar formalizações com o propósito de imitar um processo ou operação do mundo real. Desta forma, para ser realizada uma simulação é necessário construir um modelo que corresponda à situação real que se deseja simular. A simulação computacional utiliza técnicas que permitem imitar o funcionamento de praticamente qualquer tipo de operação ou processo do mundo real através da criação de modelos computacionais que os representem.

Modelos de simulação podem ser Estáticos ou Dinâmicos. Um modelo de simulação estática, é um modelo onde a passagem de tempo é irrelevante, já a modelagem dinâmica representa sistemas cujos resultados variam com a passagem do tempo.

Dependendo da natureza do sistema e das características de interesse pode se fazer uso de modelos Contínuos ou Discretos. Numa simulação contínua, as mudanças de estado ocorrem continuamente no tempo, enquanto em uma simulação discreta a ocorrência de um evento é instantânea e fixa a um ponto selecionado no tempo [Ferscha and Tripathi, 1998]. No presente trabalho sempre que for falado em simulação, estará se referindo à simulação discreta de eventos.

2.2 Simulação Distribuída

Diferente da simulação centralizada, onde um modelo é inteiramente executado em um único recurso de processamento, a simulação distribuída levou a execução de módulos da simulação para ambientes computacionais distribuídos conectados a uma rede local (LAN) ou uma rede de longa distância (WAN). Em ambos os casos a execução de um único modelo de simulação, provavelmente composto de diversos Processos Lógicos (PL), é distribuída entre os múltiplos computadores [Fujimoto, 1998].

O principal fator que leva as simulações a serem distribuídas em ambientes distintos é que ela pode reduzir o tempo de simulação drasticamente, uma vez que cada Processo Lógico possa cumprir sua função paralelamente aos outros.

A sincronização na simulação distribuída tem como objetivo geral garantir que os eventos disparados pelos Processos Lógicos ocorram em seus devidos tempos de evento de modo

ordenado. Cada evento ocorre em um instante de tempo de simulação, o tempo de evento. A ordem de execução de eventos internos de um PL utiliza como referência um tempo local de simulação, ou Local Virtual Time LVT. Já a ordem de execução de eventos externos precisa ser controlada por um tempo global reconhecido em todos os PL's do modelo, chamado de Global Virtual Time GVT [Fujimoto and Nicol, 1992].

2.3 Estados Consistentes Globais

Um estado global de um sistema de transmissão de mensagens é um conjunto dos estados individuais de todos os processos que participam deste canal de comunicação. Já um estado global consistente é definido quando este conjunto possa ser recuperado, garantindo tanto a consistência individual de cada elemento, quanto a não criação de mensagens órfãs. [Elnozahy et al., 1996].

Por exemplo, a Figura 2.1 apresenta dois estados globais: um estado consistente na Figura 2.1(a) e um estado inconsistente na Figura 2.1(b). Note que o estado consistente na Figura 2.1(a) apresenta a mensagem m1 como sendo enviada mas ainda não recebida. Este estado é considerado consistente pois representa uma situação onde a mensagem foi enviada pelo remetente e ainda está trafegando pela rede. Por outro lado, na Figura 2.1(b) o estado é inconsistente pois o processo P2 apresenta ter recebido a mensagem m2 mas o estado de P1 não apresenta o envio da mesma, caracterizando-a como uma mensagem órfã.

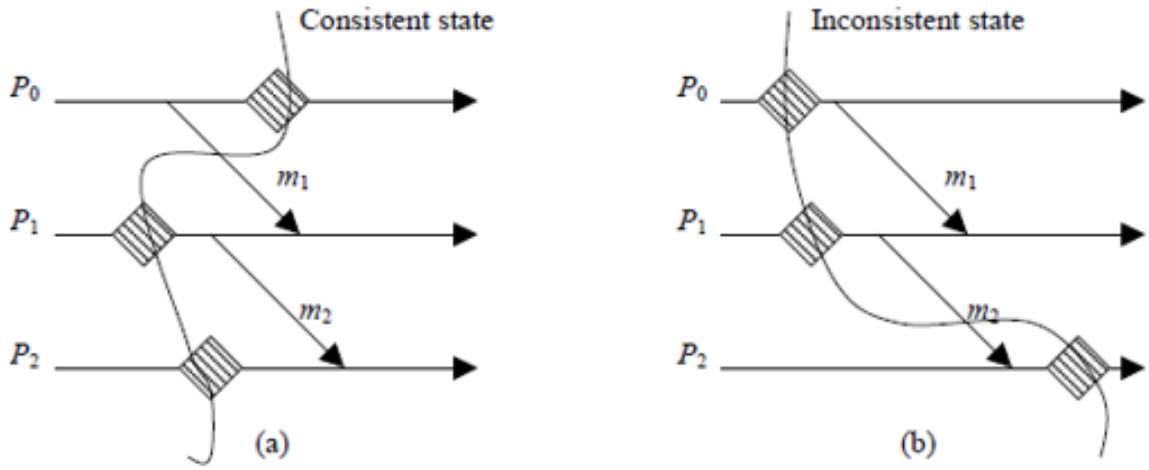

Figura 2.1: Exemplo de estado consistente e inconsistente [Elnozahy et al., 2002].

2.4 Simulação Síncrona

Na simulação síncrona, também chamada de simulação conservadora, elementos não podem retroceder no tempo no caso de alguma falha ou violação, fazendo com que o protocolo que implementa as regras de controle para a troca de mensagens garanta uma ordem segura de ocorrência de eventos durante uma simulação.

Por exemplo, um PL(A) de simulação com LVT igual a 10 deseja executar um evento interno com tempo do evento igual a 12. Contudo um PL(B) está preparando uma solicitação ao PL(A) para que ele execute um evento no tempo 11. Um algoritmo de sincronização conservadora deve garantir que o PL(A) execute o evento do tempo 12 somente depois que existam garantias que nenhum outro PL irá solicitar ao PL(A) que execute um evento num tempo menor que 12.

Para garantir a ordem de ocorrência dos eventos, na simulação síncrona, o controle geralmente é mantido pelo relógio global (GVT) [Ferscha and Tripathi, 1998]. O GVT síncrono é dado pelo maior dentre todos LVT dos PL's. Como na simulação síncrona os PL's solicitam a execução de eventos apenas para instante de tempo maiores que o GVT, garante-se que a ordem será garantida.

2.4.1 Lookahead

Uma alternativa que visa oferecer mais desempenho à simulação síncrona é a função de Lookahead. Ela permite que cada PL possa predizer quando gerará eventos no futuro, dando assim uma margem para outros processos avançarem sua computação até este intervalo de tempo. O DCB possui um mecanismo de Lookahead estático implementado, onde se estabelece um limite para este avanço.

2.4.2 Mensagem Nula

Mensagens nulas são utilizadas para evitar situações de impasse (deadlock) nos modelos síncronos. Ao enviar uma mensagem nula, um processo comunica a outro uma "previsão" sobre seu comportamento futuro, como por exemplo que não enviará nenhuma mensagem ou executará algum evento com tempo menor que o valor fornecido na mensagem nula.

2.5 Simulação Assíncrona

Também chamada de abordagem otimista, a simulação assíncrona permite que eventos possam ser executados fora de uma ordem temporal permitindo a ocorrência de violações de tempo. Entretanto, a consistência da sincronização entre os PL's é garantida através de políticas de recuperação de estados consistentes (Rollback) em situação de erro.

Tomando como exemplo a Figura 2.2 o estado de PL1 torna-se inconsistente com o envio de uma mensagem de PL0 para PL1 com o tempo de evento igual a 10. Neste caso, o PL1 deve retroceder no tempo até o LVT 10, tempo em que ocorreu a violação de tempo local, e executar novamente os eventos até o LVT 15. As mensagens enviadas a partir de PL1 entre os tempos 10 e 15, se houverem, são denominadas mensagens órfãs. Por este motivo, a recuperação de um estado seguro pode gerar um efeito conhecido como efeito dominó [Elnozahy et al., 2002].

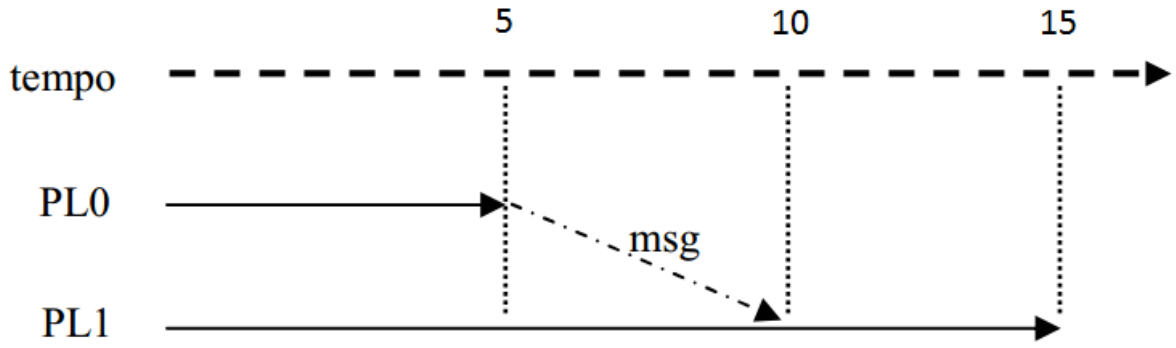

Figura 2.2: Exemplo de estado inconsistente.

Na literatura foram encontrados dois principais métodos para realizar Rollback durante uma simulação: o método de criar logs e o método de estabelecer checkpoints.

2.6 Rollback Baseado em Log

O rollback baseado em log torna explícito o fato que a execução de um processo pode ser modelada como a sequência de eventos determinísticos, tendo seu início com a execução de um evento não-determinístico. Este evento pode ser o recebimento de uma mensagem de outro processo ou um evento interno, entretanto o envio de uma mensagem é dito como um evento determinístico [Elnozahy et al., 2002].

Durante uma simulação tolerante a falhas, cada processo cria logs contendo as determi-

nantes de todos os eventos não determinísticos que ocorreram e os mantém em armazenamento estável. Caso ocorra uma falha, os processos que falharam se recuperam utilizando as determinantes dos logs para reproduzir precisamente os eventos não determinísticos da mesma maneira que estavam antes da falha.

Rollback baseados em logs podem ser divididos em três protocolos, estes serão descritos a seguir:

- Rollback pessimista baseado em log: garante que não sejam criadas mensagens órfãs ao ocorrer uma falha. Este protocolo simplifica a recuperação, coleta de lixo e entregas externas com o custo de maior overhead durante sua operação.

- Rollback otimista baseado em log: reduz o overhead de performance durante uma simulação tolerante a falhas, porém permite que mensagens órfãs sejam criadas caso haja uma falha. A possibilidade de serem criadas mensagens órfãs prejudica a recuperação, coleta de lixo e entregas externas.

- Rollback casual baseado em log: este protocolo procura combinar as vantagens de um baixo overhead de performance e rápidas entregas externas, porém exige um sistema de recuperação e coletor de lixo mais complexos.

2.7 Rollback Baseado em Checkpoint

Quando ocorre uma falha, a política de Rollback baseado em checkpoint restaura o estado do sistema de acordo com o conjunto de checkpoints. Protocolos baseados em checkpoints são menos restritivos e mais fáceis de implementar quanto aos protocolos de Log [Elnozahy et al., 2002].

As técnicas de Rollback baseadas em checkpoint podem ser classificadas em três categorias: Checkpoints coordenados, Checkpoints não-coordenados e Checkpoints induzidos a comunicação.

2.8 Checkpoints Coordenados

Checkpoints coordenados exigem que os processos orquestrem seus checkpoints a fim de formar um estado consistente global. Checkpoints coordenados simplificam a recuperação e não são suscetíveis ao efeito dominó, uma vez que todos os processos utilizarão sempre seu

último checkpoint. Com isso também supre a necessidade de coletor de lixo, uma vez que cada processo precisa de apenas um checkpoint salvo em armazenamento estável [Elnozahy et al., 2002].

Uma abordagem muito usada nos checkpoints coordenados é o bloqueio de mensagens de simulação entre os processos enquanto o protocolo de criação de checkpoints é executado. Esta abordagem causará um overhead de processamento muito alto, pois as mensagens para a coordenação entre os checkpoints podem demorar a serem transmitidas [Tamir and Sequin, 1984].

2.8.1 Coordenação de Checkpoints não Bloqueante

Um problema fundamental dos checkpoints coordenados é prevenir que processos recebam mensagens de simulação durante a criação de checkpoints, o que tornaria o checkpoint inconsistente. Considere o exemplo da Figura 2.3(a), em que o processo P0, após receber uma requisição de checkpoint do iniciador de checkpoint, envia uma mensagem m. Agora, vejamos que m chega em P1 antes que a requisição de checkpoint. Esta situação resulta em um checkpoint inconsistente, uma vez que o checkpoint $C_{1,x}$ apresenta a recepção da mensagem m de P0, enquanto o checkpoint $C_{0,x}$ não apresente a mesma sendo enviada de P0.

Se os canais de comunicação forem FIFO (First In First Out) este problema pode ser evitado fazendo com que quando os processos recebam a mensagem de requisição de checkpoint, enviem mensagens de requisição de checkpoint a todos os outros processos, como ilustra a Figura 2.3(b). Caso os canais não sejam FIFO, um marcador pode ser acoplado a todas às mensagens posteriores a um checkpoint, então, caso um processo receba uma mensagem e o marcador indica que deve ser criado um checkpoint antes da mensagem ser tratada, este checkpoint é criado, e quando a mensagem de requisição de checkpoint for recebida, ela será ignorada, como ilustra a Figura 2.3(c).

2.8.2 Confiabilidade de Comunicação em Checkpoints

Dependendo da abordagem que o canal de comunicação utiliza, os protocolos podem exigir que as mensagens sejam salvas como parte dos checkpoints. Considere o caso onde o canal de comunicação é confiável. Um processo p envia uma mensagem m depois de criar um checkpoint, e esta mensagem m alcança seu destino no processo q antes que q tenha seu checkpoint criado. Neste caso, o estado gravado do processo p deve mostrar que a mensagem m

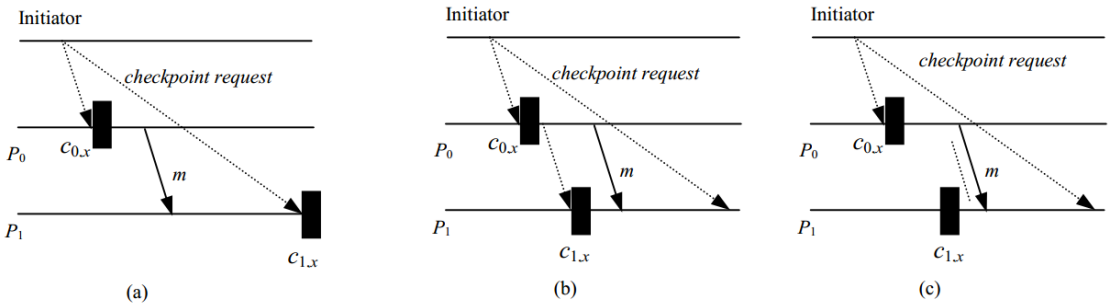

Figura 2.3: Coordenação não bloqueante de Checkpoints: (a) checkpoint inconsistente; (b) utilizando canais FIFO; (c) utilizando canais não FIFO [Elnozahy et al., 2002].

foi enviada, enquanto o estado salvo de q deve mostrar que a mensagem não foi recebida. Caso ocorra uma falha e ambos os processos forem obrigados a retroceder a estes checkpoints, seria impossível garantir a confiabilidade da entrega da mensagem m.

Para evitar este problema, o protocolo exige que todas as mensagens em trânsito sejam salvas por seu destinatário pretendido como parte de seus checkpoints. Entretanto, caso o canal de comunicação não seja confiável, não há a necessidade de gravar as mensagens em trânsito, devido ao caso que o estado salvo dos checkpoints ainda seria consistente.

2.8.3 Checkpoints Coordenados Mínimos

Checkpoints coordenados requerem que todos os processos participem em todos os checkpoints. Porém é desejado reduzir o número de processos envolvidos em uma criação de checkpoints coordenados. Isso pode ser feito desde que os processos que precisam de checkpoints novos são somente os que se comunicaram com o iniciador de checkpoint, diretamente ou indiretamente desde o último checkpoint [Koo and Toueg, 1987].

Checkpoint coordenado mínimo é um protocolo que segue duas fases. Durante a primeira fase, o iniciador de checkpoint identifica todos os processos os quais se comunicou desde o último checkpoint e os envia uma requisição. Ao receber uma requisição, o processo identifica todos os processos com que tenha se comunicado desde o último checkpoint e os envia uma requisição, e assim por diante, até que nenhum processo possa ser identificado. Na segunda fase, todos os processos identificados na primeira fase criam um checkpoint. O resultado é um estado consistente que envolve somente os processos participantes. Neste protocolo, após o recebimento de uma requisição, o processo não pode enviar mensagens a outros processos até que a segunda fase se complete, entretanto, é permitido receber mensagens de outros processos.

2.9 Checkpoints Não-Coordenados

Segundo [Elnozahy et al., 2002], checkpoints não coordenados permitem a cada processo a máxima autonomia ao decidir quando criar um checkpoint. A maior vantagem deste modelo é que cada processo cria seus checkpoints quando achar mais conveniente, assim eliminando a necessidade do envio de mensagens exclusivas para a criação de checkpoints. Porém checkpoints não coordenados possuem três desvantagens:

1. É possível ocorrer o efeito dominó, que pode causar grande perda de trabalho útil, podendo retornar até o início da computação.

2. Um processo pode criar checkpoints inúteis que nunca farão parte de estados consistentes. Checkpoints inúteis são indesejáveis pois eles causam overhead e não contribuem com o processo de Rollback.

3. Checkpoints não coordenados forçam cada processo a manter vários checkpoints, e chamar periodicamente um coletor de lixo para apagar os checkpoints que não serão mais úteis.

A fim de estabelecer estados consistentes globais, durante a recuperação, os processos gravam suas dependências em seus checkpoints enquanto operam uma simulação tolerante a falhas. Utilizando a técnica de [Bhargava e Lian 1988], temos:
Sendo $C_{i,x}$ o x-ésimo checkpoint do processo i e sendo $I_{i,x}$ o intervalo do checkpoint entre os checkpoints $C_{i,x-1}$ e $C_{i,x}$, como ilustra a Figura 2.4, se o processo P_i durante o intervalo $I_{i,x}$ envia uma mensagem m ao processo P_j, ele armazenará o par (i, x) em m. Quando P_j receber a mensagem m durante seu intervalo $I_{j,y}$, ele salvará a dependência entre $I_{i,x}$ e $I_{j,y}$, que posteriormente será armazenada quando o processo P_j criar um checkpoint.

Caso ocorra uma falha, o processo de recuperação inicia o Rollback transmitindo uma mensagem de requisição de dependências para coletar todas as dependências mantidas por cada processo. Ao receber esta mensagem, os processos paralisam sua execução e respondem com suas informações de dependências. O requisitor então calcula a linha de recuperação baseado nas informações de dependências globais e transmite mensagens de requisição de Rollback contendo a linha de recuperação. Ao receber esta mensagem, os processos cujo estado já pertence a linha de recuperação continua sua execução normalmente, caso contrário ele retrocederá para um checkpoint anterior indicado pela linha de recuperação.

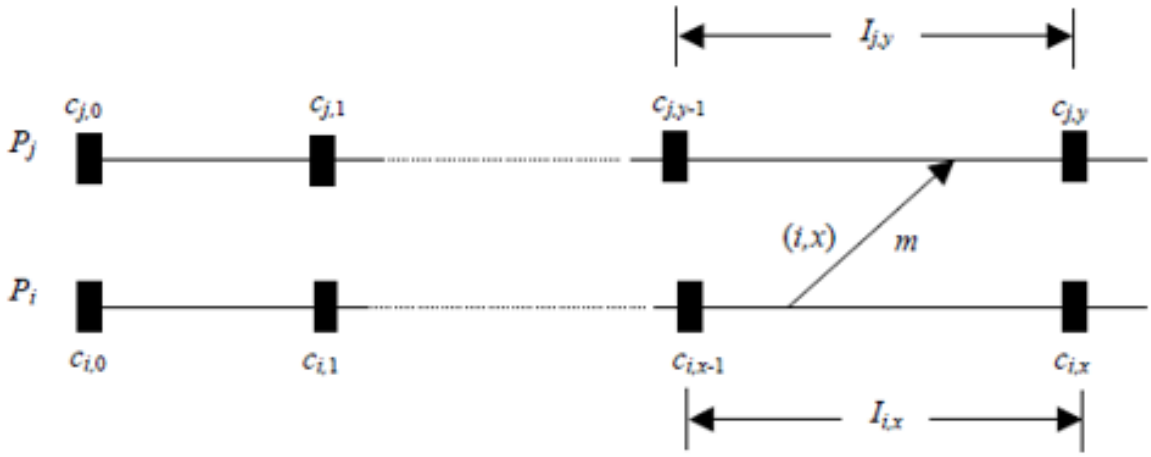

Figura 2.4: Índice de checkpoint e Intervalo de checkpoints [Elnozahy et al., 2002].

2.9.1 Anti-Mensagens

Anti-mensagens são mensagens enviadas quando ocorre uma violação de tempo [Elnozahy et al., 2002, Ferscha and Tripathi, 1998]. Estas mensagens são enviadas para os elementos que receberam mensagens num instante de tempo passado para que realizem o rollback.

2.9.2 Efeito Dominó

O efeito dominó ocorre quando um elemento recebeu uma mensagem num instante de tempo passado e o elemento que enviou a mensagem realiza a operação de rollback e volta para um tempo anterior ao envio da mensagem, tornando assim, a mensagem órfã. Com isso, o primeiro elemento deve retroceder para um tempo anterior ao recebimento desta mensagem, assim a descartando.

Por exemplo na Figura 2.5, temos uma execução sem a coordenação entre os checkpoints. Cada processo inicia a execução com um checkpoint inicial, no tempo 0. Suponha que o processo P2 falhe e retroceda ao checkpoint C. A operação de rollback “invalida” o envio da mensagem m6, assim P1 deve retroceder ao checkpoint B pois sua mensagem recebida m6 se tornou uma mensagem órfã. Retornando ao checkpoint B, a mensagem m7 também é “invalidada”, causando o mesmo efeito sobre o processo P0, que por sua vez retornará ao checkpoint A. Este rollback em cascata pode levar ao efeito dominó, reiniciando totalmente a computação.

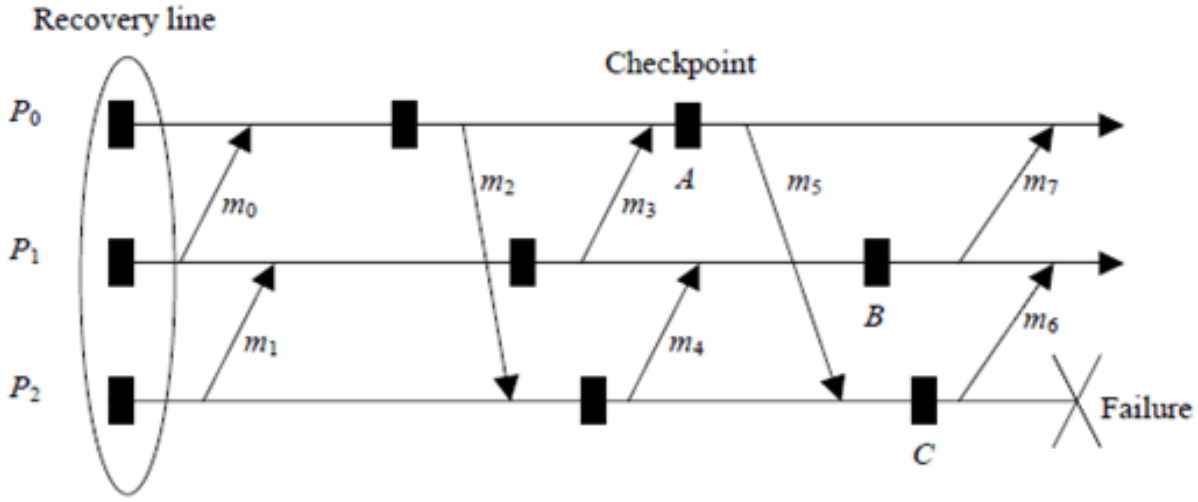

Figura 2.5: Rollback e Efeito Dominó [Elnozahy et al., 2002].

2.10 Checkpoints Induzidos a Comunicação

Protocolos de checkpoints induzidos a comunicação (CIC) evitam o efeito dominó sem a necessidade de todos os checkpoints serem coordenados. Nestes protocolos, processos criam dois tipos de checkpoints, locais e forçados. Checkpoints locais podem ser criados independentes por cada processo, enquanto os checkpoints forçados devem ser criados para garantir o progresso no caso de uma recuperação.

Oposto aos checkpoints coordenados, os protocolos CIC não trocam mensagens específicas referentes à coordenação de checkpoints para decidir quando os checkpoints forçados devem ser criados, ao invés disso, eles adicionam informações a cada mensagem da aplicação, assim o receptor pode utilizar estas informações e verificar se deve ou não criar um checkpoint forçado. Esta decisão baseia-se no receptor determinar se os padrões da comunicação passada podem levar à criação de checkpoint inútil: neste caso é criado um checkpoint forçado para quebrar estes padrões. Esta intuição foi formalizada nas noções de Z-path e Z-cycle [Netzer and Xu, 1995].

Um Z-path é uma sequência especial de mensagens que conectam dois checkpoints. Seja $\rightarrow$ a notação de Lamport: happens-before (acontece depois) [Lamport, 1978], onde o $eventoi$ $\rightarrow$ $eventoj$ se e somente se: (i) **i** e **j** pertencem ao mesmo processo e **i** acontece antes que **j**. (ii) **i** e **j** são eventos de processos diferentes e existe uma mensagem m enviada por P1 depois de P1 executar $eventoi$ que chega ao P2 antes de P2 executar $eventoj$. Agora seja $C_{i,x}$ o x-ésimo checkpoint do processo P_i. Ainda, definimos o intervalo de checkpoint sendo o período de execução de um processo entre dois checkpoints consecutivos. Dados dois checkpoints $C_{i,x}$

e $C_{j,y}$ um Z-path existe entre $C_{i,x}$ e $C_{j,y}$ se e somente se uma destas condições é assegurada:

1. $x < y$ e $i = j$; (ex: um checkpoint prcede o outro no mesmo processo)
2. Existe uma sequência de mensagens $[m0, m1, ..., m_n], n \geqslant 0$, tal que:
   - $C_{i,x} \rightarrow envio_i(m_0)$
   - $\forall\ l < n$, tanto o $recebimento_k(m_l)$ e $envio_k(m_{l+1})$, estão no mesmo intervalo de checkpoint, ou $recebimento_k(m_l) \rightarrow envio_k(m_{l+1})$; e
   - $recebimento_j(m_n) \rightarrow C_{j,y}$

Onde $envio_i$ e $recebimento_i$ são eventos de comunicação executado pelo processo P_i. Na Figura 2.6, $[m1, m2]$ e $[m3, m4]$ são exemplos de Z-path entre os checkpoints $C_{0,1}$ e $C_{2,2}$.

Um Z-cycle é um Z-path que inicia e termina no mesmo intervalo de checkpoint. Na Figura 2.6, o Z-path $[m5, m3, m4]$ é um Z-cycle contendo o checkpoint C2,2. Z-cycles são interessantes para o contexto dos checkpoints induzidos a comunicação pois pode ser provado que um checkpoint é inútil se faz parte de um Z-cycle [Netzer and Xu, 1995]. Para tanto, uma maneira de evitar checkpoints inúteis é garantir que nenhum Z-path se torne um Z-cycle.

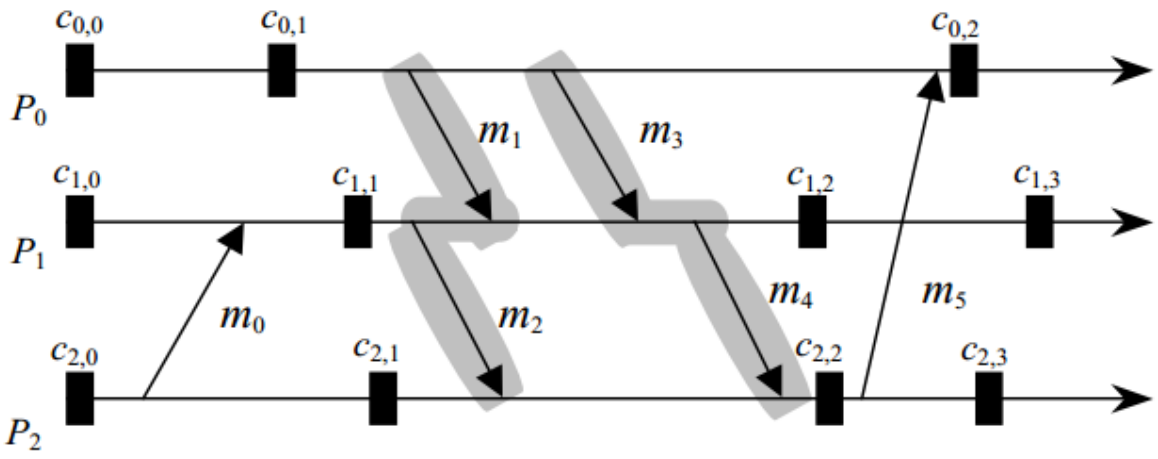

Figura 2.6: Z-Path e Z-Cycle.

2.11 High Level Architecture

O High Level Architecture (HLA) foi criado pelo Department of Defense (DoD) dos Estados Unidos da América em 1995 no âmbito de simulação interativa distribuída com base num

processo de esforço em conjunto envolvendo o governo, o ambiente acadêmico e a indústria [Dahmann et al., 1997]. Seu principal objetivo na época era aperfeiçoar o treinamento militar.

O HLA atualmente é um padrão IEEE, entretanto não exige que se siga exatamente uma linha de implementação, mas propõe mecanismos e regras que auxiliam o estudo de ambientes para simulação distribuída heterogênea.

2.11.1 Aspéctos Técnicos do HLA

O HLA trabalha com o conceito de federados, que podem ser vistos como um modelo de simulação computacional, um coletor ou visualizador de dados, ou até mesmo uma interface que permita participantes físicos na simulação, dentro do HLA, federados são representados por objetos. Ao combinar um conjunto de federados com a RTI (RunTime Interface Services – Infraestrutura de Serviços em Tempo de Execução), forma-se uma federação. A RTI é responsável pelo controle de operações de troca de mensagens entre os federados e federações [Mello, 2005].

Na Figura 2.7 temos uma representação de uma federação do HLA, nela podemos observar que os federados se comunicam com a RTI através de uma interface. Não são impostas restrições como se deve representar um federado, apenas é necessário que incorporem características que os permitam interagir com os outros objetos presentes no módulo de simulação [Dahmann et al., 1997].

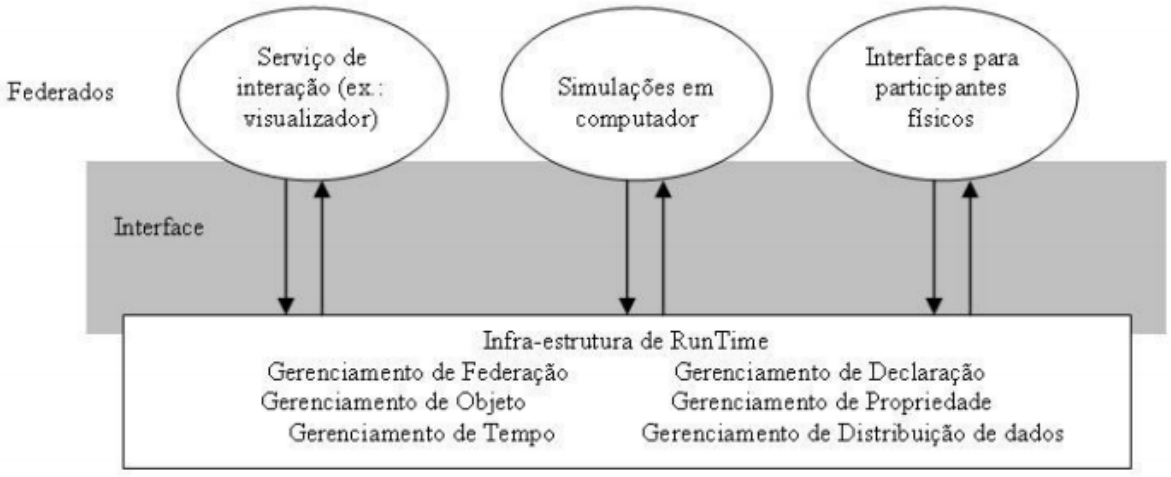

Figura 2.7: Visão de uma Federação HLA [Mello, 2005].

2.12 Distributed Co-Simulation Backbone

O DCB é uma arquitetura de simulação proposta por [Mello, 2005]. Seu propósito geral é fornecer uma estrutura que permita a simulação distribuída de modelos heterogêneos.

O DCB é composto por quatro módulos principais: o Embaixador do DCB (EDCB), o Embaixador do Federado (EF), o Núcleo do DCB (NDCB) e o gateway. Devido à heterogeneidade dos federados, o DCB utiliza o conceito de gateway para implementar uma interface de comunicação, que terá a capacidade de traduzir dados de um formato origem para um formato destino de acordo com as exigências do suporte da simulação. O gateway tem como principal tarefa o tratamento das interfaces dos componentes. Os demais tratam da sincronização, gerenciamento de dados e cooperação [Mello, 2005]. Na Figura 2.8 podemos ver a arquitetura do DCB, e perceber sua semelhança com o HLA.

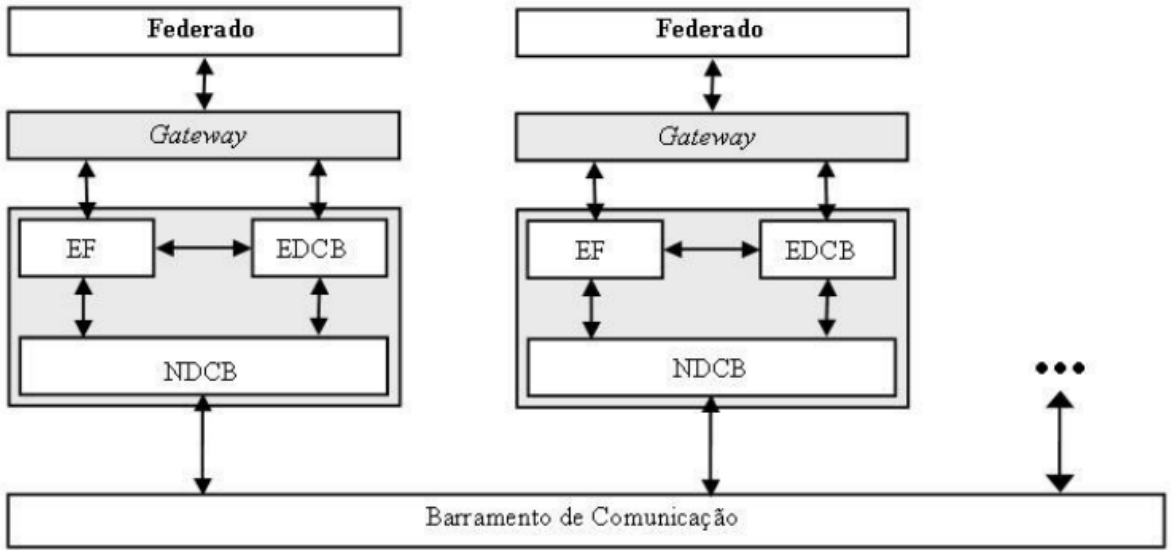

Figura 2.8: Arquitetura do DCB [Mello, 2005].

O EF tem como propósito geral o gerenciamento de mensagens recebidas de outros federados, sejam locais ou remotos. É ele quem realiza a decodificação de pacotes recebidos e participa das atividades de gerenciamento do tempo de simulação.

O EDCB tem como propósito principal o gerenciamento de mensagens emitidas pelo federado que representa. Em conjunto com o EF, também mantém o gerenciamento do tempo da simulação.

O NDCB, além de manter os serviços de comunicação por troca de mensagens, mantém um valor único em todos os nodos para o tempo virtual global (GVT).

2.12.1 Sincronização no DCB

O DCB faz a sincronização das mensagens através do tempo de evento (timestamp). O tempo de evento enviado em uma mensagem indica o tempo em que um evento deve ocorrer no federado destino [Mello, 2005]. O DCB suporta sincronização híbrida, permitindo com que cada federado avance o seu tempo local no modo assíncrono ou no modo síncrono, assim como a cooperação com federados untimed, que não utilizam tempo explicitamente no seu modelo.

No DCB, os elementos síncronos apenas poderão enviar mensagens para outros elementos síncronos se o timestamp da mensagem for maior que o GVT, para garantir a não ocorrência de violação de tempo. Elementos untimed organizam suas mensagens de acordo com a ordem em que foram recebidas. Por último os elementos assíncronos permitem o envio de mensagens com qualquer timestamp, uma vez que caso ocorra uma violação de tempo, será realizada a operação de Rollback.

3 TRABALHOS RELACIONADOS

Foram selecionados trabalhos que possuem objetivos parecidos com este trabalho [Chandy and Lamport, 1985, Cao et al., 2004] e que cooperaram para a elaboração desta proposta. Em [Chandy and Lamport, 1985], foi proposto um algoritmo que detecta estados consistentes globais a partir da troca de mensagens entre os processos. Em [Cao et al., 2004] foi proposto um algoritmo que trabalha com checkpoints não-coordenados e checkpoints coordenados.

3.1 Distributed Snapshots: Determining Global States of Distributed Systems

Em [Chandy and Lamport, 1985] foi proposto um algoritmo que um processo determina um estado global durante a computação do sistema. Os autores utilizam como exemplo de funcionamento para o algoritmo processos similares à maquinas de estado. Onde cada estado representa um processo e o canal de comunicação entre eles é representado por uma transição. O principal problema abordado foi a detecção de característica estável. Uma característica estável deve persistir: uma vez que uma característica estável se torna verdadeira ela deve continuar verdadeira. Exemplos de características estáveis são: "computação terminada" e "o sistema entrou em deadlock".

Um exemplo de uma execução pode ser observado na Figura 3.1. Nela são representados dois processos: P e Q juntamente com dois canais de comunicação, que inicialmente, estão vazios. Ao processo P enviar a mensagem M, P muda seu estado para B, o processo Q não precisa necessariamente receber esta mensagem então a mesma permanece no canal de comunicação.

Os algoritmos propostos em [Chandy and Lamport, 1985] funcionam da seguinte maneira:

Algoritmo 1: Regra de Envio do Marcador para um processo p

```
1 Para cada canal de comunicação c, originado deste e
  direcionado para fora a partir de p;
2 p envia uma marca ao longo de c após p armazenar seu
  estado e antes de p enviar outras mensagens em c.
```

As regras de envio e recebimento de marcadores garantem que se uma marca for recebida por um canal, o processo armazenará seu estado e o estado de seus canais de comunicação de entrada. Para garantir que o registro do estado global termine em tempo finito, cada processo

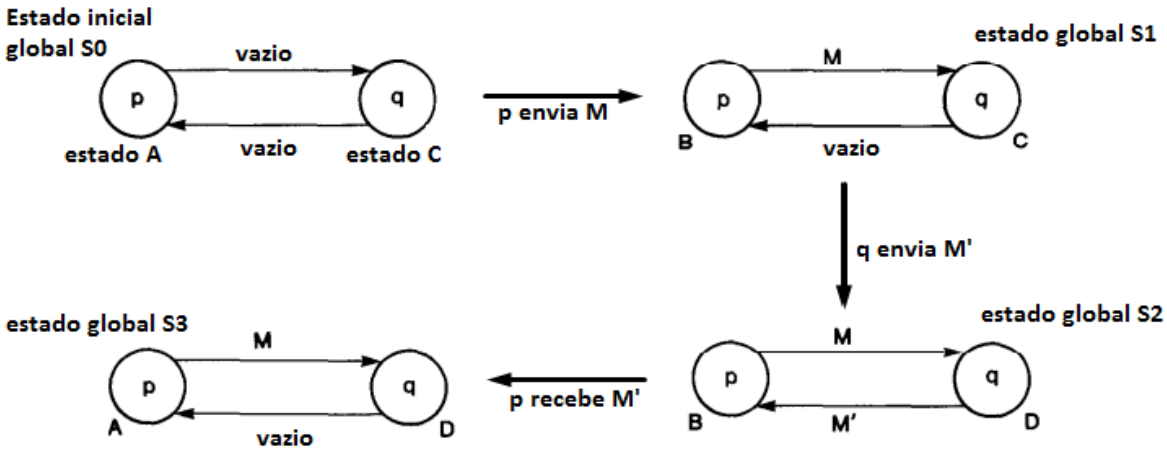

Figura 3.1: Exemplo de execução [Chandy and Lamport, 1985].

Algoritmo 2: Regra de Recebimento do Marcador para um processo q

```
1 Ao receber um marcador por seu canal c;
2 se q não armazenou seu estado então
3 |  q armazena seu estado;
4 |  q armazena o estado de c como uma sequência vazia;
5 else
6 |  q armazena o estado de c de acordo com a sequência de
  |  mensagens recebidas ao longo de c após que o estado
  |  de q foi salvo e antes que recebeu o marcador por c.
7 end
```

deve garantir que nenhum marcador permaneça para sempre em um canal de comunicação incidente e que grave seu estado em tempo finito a partir do início do algoritmo.

Este trabalho pode ser útil para o DCB pois se trata de uma abordagem não-coordenada de criação de checkpoints e é possível adaptá-la ao DCB.

3.2 Checkpointing in Hybrid Distributed Systems

Em [Cao et al., 2004] foi proposto um algoritmo que garante a criação de estados consistentes a partir da utilização de checkpoints não-coordenados (locais) e checkpoints coordenados (forçados). No trabalho, é definido como *IN* os processos com checkpoints independentes e *CO* é utilizado para os processos com checkpoints coordenados.

Uma visão geral do algoritmo criado pode ser descrita em duas regras:

Regra 1: IN cria um novo checkpoint coordenado caso envie uma mensagem M para qualquer CO. Ou seja, IN cria um checkpoint coordenado imediatamente depois de enviar M para algum CO.

Regra 2: P_i IN cria um novo checkpoint não-coordenado antes de receber uma mensagem M de algum CO apenas se P_i enviou alguma mensagem depois de seu último checkpoint.

Algoritmos desenvolvidos:

Algoritmo 3: Implementação da Regra 1

```
1 se M é destinada a algum CO então
2 |  Bloqueie o estado de IN;
3 |  S = estado atual de IN + "M foi enviada";
4 |  Armazene S em armazenamento estável;
5 |  Envie M para CO;
6 |  Registre S como um checkpoint coordenado;
7 |  Desbloqueie o estado de IN;
8 fim
```

Algoritmo 4: Implementação da Regra 2

```
1 se M é enviada de algum CO e P_i enviou alguma mensagem durante seu intervalo
  de checkpoint atual antes do recebimento de M então
2 |  P_i cria um novo checkpoint IN;
3 fim
4 Registre o recebimento de M;
5 Processe M;
```

Segundo seus atores, este algoritmo possui diversas vantagens, sendo elas: Fácil implementação; Apenas os esquemas de checkpoint necessitam ser modificados, uma vez que não interfere em outra parte do sistema; Relativamente baixo overhead de processamento para os checkpoints Coordenados; Pode ser aplicada a qualquer momento da computação sem necessitar de mudanças na aplicação.

Porém o fato de apresentarem um algoritmo de coletor de lixo (o qual não foi discutido no presente trabalho), mostra que possivelmente há a criação de checkpoints inúteis, além de que bloquear um elemento (como visto no Algoritmo 3) não é algo desejável. Por outro lado o estudo deste trabalho ainda pode ser útil para a implementação no DCB pois mesmo bloqueando os processos, a criação coordenada age somente de forma local e não envia mensagens específicas para sua criação.

4 METODOLOGIA

Na primeira etapa do projeto foi definido o tema de pesquisa, e a partir do tema e de dois trabalhos principais, que são [Elnozahy et al., 2002] e [Mello, 2005], foi realizado uma pesquisa no Scholar Google pela string de busca *"Consistent States Checkpoint in Distributed Simulation"* combinado a verificação das referências destes dois trabalhos. Do resultado inicial da pesquisa, foram considerados os mais relevantes: mais citados e títulos mais próximos do trabalho proposto. Durante a leitura dos trabalhos selecionados, foram levantadas as técnicas utilizadas para identificação de estados consistentes e foram adicionados artigos relevantes ao conjunto inicial.

Após o estudo, foi decidido trabalhar com as abordagens de checkpoints não-coordenados e checkpoints induzidos a comunicação, pelas suas qualidades descritas nos trabalhos.

O projeto será dividido em 4 etapas: (i) Identificação de estados seguros em sistemas distribuídos. (ii) Criação de checkpoints através de estados seguros. (iii) Implementação da criação de Checkpoints no DCB. (iv) Criação de um estudo de caso para validação dos resultados.

Na primeira etapa será feito um estudo sobre as técnicas existentes que permitem a identificação de estados seguros em sistemas distribuídos. Também será necessário verificar quais destas técnicas correspondem com as características do DCB.

Na segunda etapa se iniciará o estudo de integração da técnica de identificação de estados seguros com a criação de checkpoints, também se fazendo necessário a verificação junto as características do DCB de como integrar estas técnicas.

A terceira etapa será a implementação da técnica especificada nas etapas i e ii no DCB. O DCB foi desenvolvido na linguagem JAVA, então a mesma será usada neste trabalho.

Na última etapa será realizada a validação dos resultados obtidos por meio de um estudo de caso. Este estudo de caso será feito botando em prática uma simulação tolerante a falhas com elementos assíncronos no DCB e verificando se o seu resultado está de acordo com o esperado.

4.1 Cronograma e Orçamento

A Tabela 4.1 apresenta o cronograma de trabalho do projeto e da monografia, a Tabela 4.2 apresenta o orçamento do projeto.

Atividades	**Nov**		**Dez**		**Jan**		**Fev**		**Mar**		**Abr**		**Mai**		**Jun**		**Jul**	
Execução da etapa I		X	X	X	X	X	X											
Execução da etapa II				X	X	X	X	X	X	X								
Execução da etapa III						X	X	X	X	X	X	X						
Execução da etapa IV								X	X	X	X	X	X	X				
Redação da Monografia										X	X	X	X	X	X	X	X	

Tabela 4.1: Cronograma de Ação

Recurso	**Quantidade**	**Valor(Uni.)**	**Observação**
Livros	0	R$ 0,00	Disponíveis na biblioteca da UFFS
Artigos	0	R$ 0,00	Bibliotecas online e Periódicos assinados pela UFFS
Equipamentos	0	R$ 0,00	Computador pessoal
Correção Ortográfica	1	R$ 150,00	
Impressão Projeto	3	R$ 15,00	
Encadernação Projeto	3	R$ 6,00	
Impressão Monografia	3	R$ 20,00	
Encadernação Monografia	3	R$ 6,00	
Total		R$ 291,00	

Tabela 4.2: Orçamento para execução do projeto

REFERÊNCIAS

[1] J. Cao, Y. Chen, K. Zhang, and Y. He. Checkpointing in hybrid distributed systems. In *Parallel Architectures, Algorithms and Networks, 2004. Proceedings. 7th International Symposium on*, pages 136–141. IEEE, 2004.

[2] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 3(1):63–75, 1985.

[3] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly. The department of defense high level architecture. In *Proceedings of the 29th conference on Winter simulation*, pages 142–149. IEEE Computer Society, 1997.

[4] E. Elnozahy, D. Johnson, and Y. Wang. A survey of rollback-recovery protocols in message-passing systems. Technical report, Technical Report CMU-CS-96-181, Carnegie Mellon Univ, 1996.

[5] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)*, 34(3):375–408, 2002.

[6] A. Ferscha and S. K. Tripathi. Parallel and distributed simulation of discrete event systems. 1998.

[7] R. Fujimoto and D. Nicol. State of the art in parallel simulation. In *Proceedings of the 24th conference on Winter simulation*, pages 246–254. ACM, 1992.

[8] R. M. Fujimoto. Time management in the high level architecture. *Simulation*, 71(6): 388–400, 1998.

[9] R. Koo and S. Toueg. Checkpointing and rollback-recovery for distributed systems. *Software Engineering, IEEE Transactions on*, (1):23–31, 1987.

[10] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[11] B. A. d. Mello. Co-simulação distribuída de sistemas heterogêneos. page 145, 2005.

[12] R. H. Netzer and J. Xu. Necessary and sufficient conditions for consistent global snapshots. *IEEE Transactions on Parallel & Distributed Systems*, (2):165–169, 1995.

[13] P. F. Reynolds Jr. Heterogenous distributed simulation. In *Proceedings of the 20th conference on Winter simulation*, pages 206–209. ACM, 1988.

[14] Y. Tamir and C. H. Sequin. Error recovery in multicomputers using global checkpoints. In *In 1984 International Conference on Parallel Processing*. Citeseer, 1984.