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

LIMITAÇÃO DO ESPAÇO DE ARMAZENAMENTO DE *CHECKPOINTS* EM SIMULAÇÃO DISTRIBUÍDA OTIMISTA

MAICON GHIDOLIN

CHAPECÓ
2017

MAICON GHIDOLIN

LIMITAÇÃO DO ESPAÇO DE ARMAZENAMENTO DE *CHECKPOINTS* EM SIMULAÇÃO DISTRIBUÍDA OTIMISTA

Trabalho de conclusão de curso de graduação apresentado como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Prof. Dr. Braulio Adriano de Mello

Co-orientador: Prof. Ricardo Parizotto

CHAPECÓ
2017

MAICON GHIDOLIN

LIMITAÇÃO DO ESPAÇO DE ARMAZENAMENTO DE *CHECKPOINTS* EM SIMULAÇÃO DISTRIBUÍDA OTIMISTA

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

Aprovado em: ___________

BANCA EXAMINADORA:

_____
Dr. Braulio Adriano de Mello - UFFS

_____
Ricardo Parizotto - UFFS

_____
Dr. Claunir Pavan - UFFS

_____
Dr. Emílio Wuerges - UFFS

Let's go invent tomorrow instead of worrying about what happened yesterday
— STEVE JOBS

RESUMO

Modelos de simulação distribuída que possui componentes assíncronos estão sujeitos a retrocesso no tempo causados por violação de tempo. *Checkpoints* são utilizados para restauração de estados consistentes anteriores a violação de tempo. Ao longo da simulação os componentes podem gerar *checkpoints* não úteis em caso de retrocesso. Quanto maior a simulação for, maior será a quantidade de *checkpoints* desnecessários que não contribuem para a recuperação da simulação em caso de falhas e ocasionam desperdício de processamento e memória. O objetivo deste trabalho é apresentar algumas técnicas de limitação do espaço de armazenamento de *checkpoints* e algumas métricas para indicar quando um *checkpoint* pode ser substituído ou removido sem prejuízo para a simulação. As técnicas e métricas serão implementadas no DCB (*Distributed Co-simulation Backbone*) a fim de fazer experimentos para avaliar o desempenho e a utilidade dos resultados de acordo com o objetivo proposto.

Palavras-chave: Simulação distribuída, Componentes assíncronos, Checkpoints, Memória limitada.

ABSTRACT

Distributed simulation models that have asynchronous components are subject to time backtracking caused by time violation. *Checkpoints* are used for restoring consistent states prior to time violation. Throughout the simulation the components can generate non-useful *checkpoints* in case of throwback. The bigger the simulation is, the greater the number of unnecessary *checkpoints* that do not contribute to simulation recovery in case of failures and result in wasted processing and memory. The objective of this work is to present some techniques of limiting the storage space of *checkpoints* and some metrics to indicate when a *checkpoint* can be replaced or removed without prejudice to the simulation. The techniques and metrics will be implemented in Distributed Co-simulation Backbone (DCB) in order to perform experiments to evaluate the performance and usefulness of the results according to the proposed objective.

Keywords: Distributed simulation, Asynchronous components, Checkpoints, Limited memory.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

DCB	*Distributed Co-Simulation Backbone*
DCBK	*Distributed Co-Simulation Backbone Kernel*
DCBR	*Distributed Co-Simulation Backbone Receiver*
DCBS	*Distributed Co-Simulation Backbone Sender*
GVT	*Global Virtual Time*
LAN	*Local Area Network*
LCC	*Local Causality Constraint*
LVT	*Local Virtual Time*
PL	*Processo Lógico*
WAN	*Wide Area Network*

SUMÁRIO

1 INTRODUÇÃO

1.1 Tema

Este trabalho aborda estratégias de gerenciamento de *checkpoints* em simulações distribuídas com componentes assíncronos.

1.2 Contextualização

A ideia e propósitos principais da simulação computacional é representar o comportamento de um sistema real através de modelos que imitam suas caraterísticas e propriedades.

A adoção de técnicas de simulação se torna uma alternativa que viabiliza experimentos em sistemas perigosos ou de alto custo, onde testes em situações reais seriam impraticáveis. Além disso, métodos de simulação auxiliam no entendimento de sistemas existentes, na análise de novos sistemas antes de sua implementação e na análise de performance de sistemas existentes em condições variadas (LAW; KELTON, 1997).

Uma grande simulação pode ser dividida em sub-simulações que podem ser executadas simultaneamente, reduzindo assim o tempo de execução (FUJIMOTO, 2000). Estas sub-simulações (ou componentes) podem ser executas em plataformas multiprocessadas, em computadores conectados por uma rede local (LAN) ou em computadores geograficamente distribuídos, conectados via *Wide Area Network (WAN)* (FUJIMOTO, 2001).

Quando componentes são executados distribuidamente, políticas de gerenciamento de tempo se tornam necessárias, uma vez que o tempo não é mais controlado por uma única máquina. O gerenciamento de tempo é responsável por garantir a sincronização da simulação, ou seja, garantir que os eventos ocorram na ordem correta (FUJIMOTO, 2001). Eventos são ações que alteram o estado do modelo. Para isso, cada componente controla seu tempo local (LVT - *Local Virtual Time*) e o tempo global (GVT - *Global Virtual Time*) é calculado pelo maior LVT para simulações síncronas e pelo menor LVT para simulações assíncronas.

Uma simulação é dita síncrona (conservadora) quando não aceita violações de tempo, ou seja, um processo somente executa um evento quando tiver a garantia que nenhum outro evento com *timestamp* menor vai ser solicitado (FUJIMOTO, 2001). O *timestamp* é uma informação adicionada na mensagem que solicita a execução de um evento e informa ao componente de destino qual é o tempo que o evento deve ser executado.

Ao contrário de uma simulação síncrona, uma simulação é dita assíncrona (otimista) quando permite que violações de tempo (LCC) ocorram, ou seja, quando permite que um componente solicite a execução de um evento com *timestamp* menor que o LVT do componente de destino. Neste caso o componente de destino é forçado a refazer a simulação a partir de um ponto livre de erros (estado seguro) (QUAGLIA, 1999) (FUJIMOTO, 2001).

1.2.1 Definição do problema

A criação de *checkpoints* é uma forma de suporte à restauração de estados seguros na ocorrência de violações de tempo. Um processo salva periodicamente seu estado para que, em caso de falha, possa retornar para o último estado global consistente (CAO; SINGHAL, 1998). Entretanto, o processo de salvamento de *checkpoints* consome recursos de armazenamento (ELNOZAHY et al., 2002) e consequentemente o salvamento com espaço de endereçamento ilimitado desses *checkpoints* pode acarretar em uma falta de memória, principalmente em simulações grandes, podendo causar até a sua interrupção. Embora existam estudos que proponham métricas para reduzir a criação de *checkpoints* inúteis (QUAGLIA, 1999), em geral essas técnicas não incorporam mecanismos que limitam o armazenamento de *checkpoints*. Um exemplo disso é o DCB, que é uma arquitetura de suporte a simulação distribuída de sistemas heterogêneos (MELLO, 2005).

Isso acontece pois há uma preferência pelo uso de políticas de recuperação sob demanda como o *garbage collection*, o *fossil collection* e protocolos baseados em rollbacks (*rollback-based protocols*) ao invés de mecanismos que limitam o armazenamento de *checkpoints*.

Uma abordagem comum de *garbage collection* é identificar o último conjunto consistente de *checkpoints* e remover todas as informações de eventos que ocorreram antes disso (WANG, 1993). Entretanto em casos especiais (i.e., quando o último estado consistente é o estado inicial), essa abordagem não conseguirá remover nenhum *checkpoint*, levando a um transbordamento de memória (LIU; CHEN, 1999).

O *fossil collection* tem uma abordagem parecida com o *garbage collection*. A implementação mais simples compara o *timestamp* da informação com o valor do GVT. Assim toda informação ocorrida antes do GVT pode ser removida (YOUNG; WILSEY, 1996). Esse método tem o mesmo problema que o *garbage collection*, correndo o risco de não conseguir excluir nenhum *checkpoint*.

Quando as abordagens acima não conseguem recuperar mais nada e o sistema fica sem

memória, os protocolos baseados em *rollbacks* podem ser invocados. Estes protocolos induzem os processos a executarem *rollbacks*, gerando informações inúteis, tais como *logs* de mensagens e *checkpoints* obsoletos, que podem ser apagadas liberando a memória. Porém, se essa abordagem também não liberar memória, o programa é terminado (DAS; FUJIMOTO, 1994).

Ao contrário de simulações que utilizam políticas de recuperação sob demanda, que ainda correm o risco de ficar sem memória, simulações que limitam o espaço de memória (i.e., uso de memória virtual) passam uma impressão de memória infinita, devido as políticas de substituição (similares a protocolos de memória cache em sistemas operacionais) (MITTRA, 1995).

A dificuldade está justamente em criar/adaptar esses mecanismos que limitem o espaço de memória utilizado pelos *checkpoints* de cada processo e encontrar técnicas de substituição desses *checkpoints*, para quando o espaço de memória definido atingir um estado crítico. Geralmente é complexo implementar soluções desse tipo que não tenham impacto negativo no desempenho da simulação.

1.3 Objetivos

1.3.1 Geral

Especificar e validar políticas para o salvamento e substituição de *checkpoints* em um espaço de memória finito em tempo de simulação.

1.3.2 Específicos

- Com base na análise de técnicas já existentes de gerenciamento de memória limitada para armazenamento de *checkpoints* em simulações distribuídas, identificar quais delas apresentam o melhor custo-benefício
- Através do estudo de políticas de substituição em *buffers*, memória cache e memória virtual, classificar quais políticas podem ser adaptadas para um ambiente de simulação
- Criar ou adaptar uma política de substituição para controlar um espaço de memória limitado para armazenamento de *checkpoints* em simulação distribuída
- Identificar uma estratégia para estimar o tempo de simulação em que o espaço de armazenamento de *checkpoints* se torna insuficiente com base na frequência de ocorrência de

checkpoints e no tamanho do espaço de armazenamento.

1.4 Justificativa

Modelos grandes ou complexos requerem uma grande disponibilidade de processamento e memória para que a simulação seja executada em um tempo aceitável. A distribuição dessa simulação além de aumentar o poder de processamento e o montante de memória disponível, também permite reduzir o tempo da simulação. Entretanto, a distribuição também aumenta a complexidade no controle da simulação e exige que existam algoritmos de recuperação de falhas, como por exemplo, a violação de tempo.

A troca de informações entre os processos auxilia na identificação de pontos para recuperação (*checkpoints*). Esses *checkpoints* e informações trocadas pelos processos consomem recursos de armazenamento. Com a execução da simulação mais informações serão armazenadas, gerando um conjunto de informações que tornam-se desnecessárias, por exemplo, quando são anteriores ao último estado seguro da simulação (ELNOZAHY et al., 2002). Mesmo que existam métodos que recuperam o armazenamento usado por informações desnecessárias, estes não impedem um *overflow* da memória. Por isso, políticas para gerenciar espaços limitados de armazenamento de *checkpoints* são úteis.

2 REFERENCIAL TEÓRICO

2.1 Simulação

Simulação é uma maneira de imitar um sistema real através de modelos. Ou seja, através de recursos computacionais é possível representar um sistema e executar experimentos a fim de entender como um sistema funciona ou deveria funcionar, ajudando na tomada de decisões (PIDD, 1994).

Dentre as vantagens dessa abordagem estão: A possibilidade de verificar o sistema antes de implementar, entender como um sistema funciona, testar novas regras ou hipóteses, identificar erros ou gargalos, analisar sistemas de longa duração em um tempo curto e fazer testes em sistemas caros ou perigosos (SHANNON, 1998) (LAW; KELTON, 1997).

Quando uma simulação se torna grande ou complexa demais, técnicas de simulação distribuída podem sem utilizadas, subdividindo uma simulação em vários processos paralelos ou distribuídos.

2.2 Simulação Distribuída

Uma simulação pode ter seus componentes subdivididos em vários processos que serão executados paralelamente, assim, sendo nomeada de simulação distribuída. Como vários componentes vão executar ao mesmo tempo isso acaba diminuindo o tempo da simulação. Além disso, a distribuição da simulação também permite a heterogeneidade dos componentes (*p.ex.* máquinas de diferentes arquiteturas) (FUJIMOTO, 2001).

Como cada componente é um processo, eles interagem entre si por troca de mensagens. Visto que a simulação não é mais executada em uma única máquina, cada componente possui uma noção de tempo local (*LVT - Local Virtual Time*) para que seus eventos sejam sincronizados com os eventos de outros componentes, garantindo uma ordem correta de execução (PREISS; MACINTYRE; LOUCKS, 1992). Esses tempos locais dos componentes também auxiliam no cálculo do tempo virtual global (*GVT - Global Virtual Time*).

Se tratando de sincronização de eventos há duas abordagens: a síncrona (conservadora) e a assíncrona (otimista).

2.2.1 Simulação Síncrona (Conservadora)

Em uma abordagem conservadora o algoritmo de sincronização deve tomar cuidado para que violações de tempo (*LCC - Local Constraint Causality*) não ocorram. Uma violações de tempo (*LCC*) ocorre quando um componente recebe um evento com tempo menor que o seu tempo atual (*LVT*). Portanto, o algoritmo deve determinar quando é seguro um componente processar um evento, garantindo que nenhum evento com tempo menor será recebido depois que o componente avançou no tempo (FUJIMOTO, 2001).

Nessa estratégia os processos ficam ociosos, devido a sincronia com os outros processos, levando a uma perda de eficiência (FERSCHA; TRIPATHI, 1994).

2.2.2 Simulação Assíncrona (Otimista)

A abordagem assíncrona também é conhecida como otimista, pois avança no tempo sem se preocupar com os eventos solicitados pelos demais processos.

Essa estratégia permite que violações de tempo (*LCC*) ocorram, tornando a simulação inconsistente. Portando, para esses métodos se tornam necessários algoritmos de recuperação de falhas.

Na figura 2.1 é ilustrada uma violação de tempo (*LCC*) onde o processo PL0 solicita a execução de um evento por PL1 com um tempo menor (40) que o atual de PL1 (60), criando uma inconsistência na simulação.

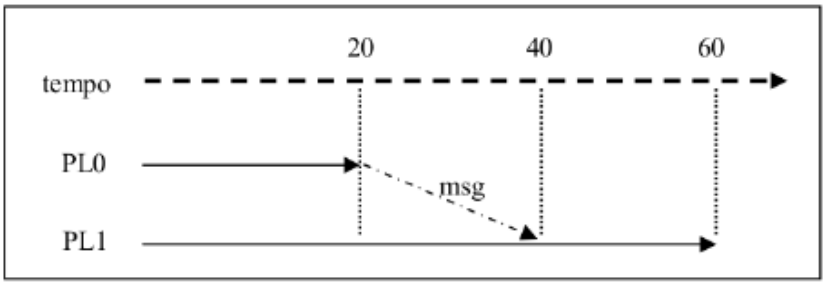

Figura 2.1 – LCC - Violação de tempo

2.3 Gerenciamento de Falhas

Nas simulações assíncronas são necessários algoritmos de recuperação de falhas. Esses algoritmos, chamados de protocolos de *rollback*, são responsáveis por retroceder uma simulação

à um estado global consistente (livre de falhas), não sendo necessário iniciar a simulação do começo.

O estado global é um conjunto dos estados de cada processo participante do sistema (ELNOZAHY et al., 2002). Um estado global é dito consistente se não existem mensagens órfãs. Uma mensagem é órfã quando o evento que recebeu está salvo, mas o evento que enviou se perdeu (KUMAR; CHAUHAN; KUMAR, 2010).

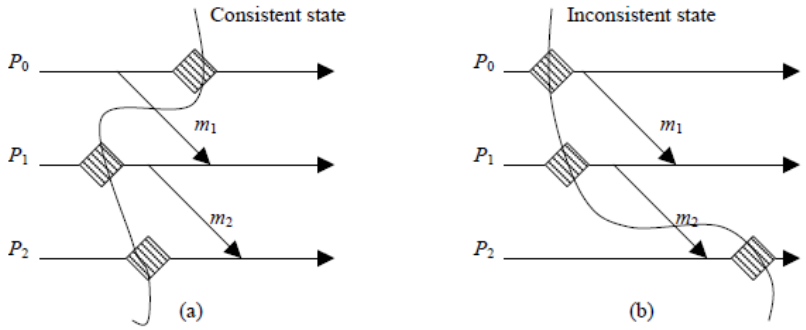

Figura 2.2 – Estado global consistente e inconsistente (ELNOZAHY et al., 2002)

A figura 2.2 ilustra os estados globais. Em 2.2 (a) o estado é consistente e em 2.2 (b) o estado é inconsistente, pois a mensagem *m2* é órfã.

2.4 *Checkpoints*

Checkpointing é uma técnica bem sucedida usada para recuperação de falhas em sistemas distribuídos (KUMAR; CHAUHAN; KUMAR, 2010).

Um *checkpoint* é uma cópia do estado da aplicação que pode ser usado para reiniciar a execução da simulação em caso de falhas (SATO et al., 2012). Caso uma violação de tempo ocorra, um *rollback* restaura a simulação para o estado consistente mais recente.

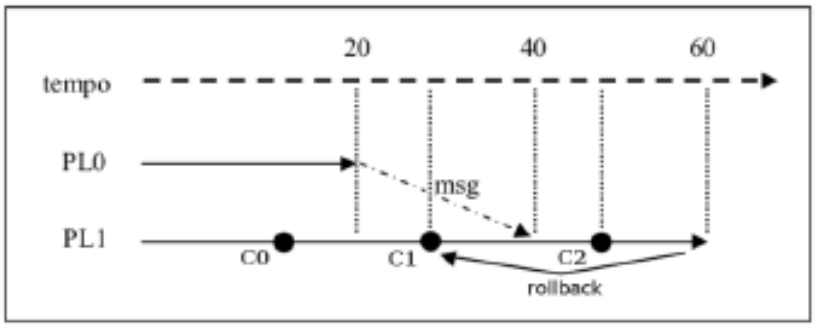

Figura 2.3 – Exemplo de *checkpointing*

A figura 2.3 mostra um exemplo da utilização de *checkpoints*. O processo PL0 envia uma mensagem com tempo menor (40) que o atual de PL1 (60) causando uma violação de tempo e forçando um *rollback* para o *checkpoint* C1 que possui tempo anterior da mensagem enviada por PL0, garantindo a ordem correta da execução.

Técnicas de *rollback* baseadas em *checkpoints* podem ser classificadas como *checkpoints* coordenados, não-coordenados e induzidos por comunicação.

Checkpoints coordenados sincronizam a criação de *checkpoints*, ou seja, quando um processo cria um *checkpoint* ele solicita para que todos os processos relevantes criem *checkpoints* também (CAO; SINGHAL, 1998). A desvantagem dessa abordagem é a grande quantidade de mensagens de controle trocadas.

A não-coordenação de *checkpoints* permite que cada processo decida quando tomar *checkpoints*. A vantagem dessa técnica é a diminuição do *overhead* causado pela troca de mensagens, visto que cada processo toma *checkpoints* quando for mais conveniente, sem depender dos outros processos. Entre as desvantagens dessa abordagem estão a possibilidade de efeito dominó em caso de *rollback* e a criação de *checkpoints* inúteis (ELNOZAHY et al., 2002).

Checkpoints induzidos por comunicação combinam as duas técnicas citadas acima. Nesse protocolo os processos criam dois tipos de *checkpoints*, locais e forçados. *Checkpoints* locais podem ser criados independentemente pelos processos, enquanto os *checkpoints* forçados são criados baseados nas informações trocadas por mensagens entre os processos. Essa abordagem diminui o *overhead* criado por mensagens de controle e evita o efeito dominó (ELNOZAHY et al., 2002).

2.5 *Checkpoints* Inúteis

Quando os componentes criam *checkpoints* de forma não-coordenada, pode ser que alguns desses *checkpoints* nunca serão usados, ou seja, não farão parte de um estado global consistente, sendo considerados inúteis.

Em (PARIZOTTO; MELLO, 2017) são observados três tipos de *checkpoints* inúteis:

- *Checkpoints* inalcançáveis: Nunca são restaurados por operações de *rollback* (Figura 2.4 C0).
- *Checkpoints* insuficientes: *Checkpoints* que são restaurados, mas não fazem parte de um estado global consistente (Figura 2.4 C1).

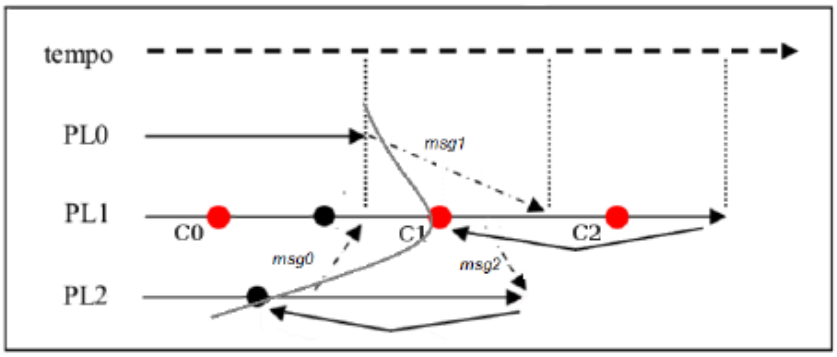

Figura 2.4 – Exemplo de *checkpoints* inúteis

- *Checkpoints* inconsistentes: *Checkpoints* que armazenam um estado com *LVT* maior que o *LVT* atual do processo (Figura 2.4 C2).

2.6 Eliminação de *Checkpoints*

Grandes simulações exigirão uma grande quantidade de *checkpoints* para controle de falhas. Com o andamento da simulação alguns *checkpoints* podem se tornar obsoletos, ou seja, não serão mais úteis para uma execução de *rollback*. *Checkpoints* obsoletos (inúteis) se tornam um problema, pois além de não contribuírem para a recuperação da simulação em caso de falhas, eles consomem espaço de armazenamento.

A partir deste problema surgem os algoritmos de recuperação de memória sob demanda, que quando executados recuperam a memória ocupada por informações inúteis, tais como *checkpoints* e *logs* de mensagens. Entre os mais conhecidos algoritmos de recuperação estão o *Garbage Collection* e o *Fossil Collection*. O capítulo dos trabalhos relacionados comenta o funcionamento deles.

2.7 DCB

O DCB é uma arquitetura de simulação onde é possível criar e executar simulações de modelos distribuídos e heterogêneos.

O DCB é composto por quatro módulos principais: o *gateway*, o Núcleo do DCB (*DCB Kernel - DCBK*), o Expedidor do DCB (*DCB Sender - DCBS*) e o Receptor do DCB (*DCB Receiver - DCBR*).

Devido a heterogeneidade dos componentes, é necessário uma interface de comunicação entre o elemento e os demais módulos. No DCB o responsável por fazer essa interface é o

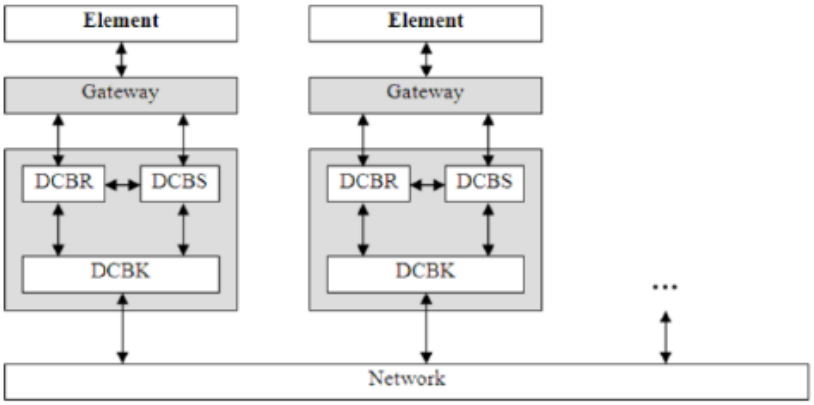

Figura 2.5 – Arquitetura do DCB

gateway. Ou seja, o *gateway* é capaz de traduzir as informações de um formato de origem para um formato de destino de acordo com a necessidade.

O núcleo do DCB (*DCBK*) é o serviço que controla a troca de mensagens. Além disso, ele mantém um único valor do tempo virtual global (*GVT*) para todos os nodos.

O *DCBR* é responsável pelas mensagens recebidas de outros componentes. Ele decodifica as mensagens recebidas e participa do gerenciamento de tempo e simulação.

O *DCBS* é o serviço de gerenciamento de mensagens enviadas pelo componente. Em conjunto com o *DCBR*, também participa do gerenciamento de tempo.

A sincronização das mensagens no DCB é feita através do tempo de evento (*timestamp*). O *timestamp* é enviado junto da mensagem e indica a ordem que o evento deve ser executado.

O DCB permite que os componentes avancem no tempo de modo assíncrono ou síncrono, suportando assim um modelo de sincronização híbrido.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta um estudo de métodos propostos para reduzir a média de *checkpoints* criados em simulações distribuídas e técnicas de recuperação sob demanda de espaço ocupado por informações desnecessárias, tais como *logs* de mensagens e *checkpoints* obsoletos. Também serão discutidas algumas estratégias existentes de gerenciamento de memória limitada e alguns benefícios na utilização destes métodos.

Em (WANG, 1993) uma abordagem comum do *gabage collection* é citada usando a ideia de *checkpoints* obsoletos, onde o mais recente conjunto de *checkpoints* consistente é identificado e todas as informações referentes a eventos que ocorreram antes disso são descartadas. A vantagem dessa abordagem é a facilidade de implementação.

Liu e Chen em (LIU; CHEN, 1999) encontram um problema no algoritmo citado por Wang *et al.*, provando facilmente que em casos especiais a abordagem comum do *garbage collection* não conseguirá apagar nenhum checkpoint. Um caso especial citado é o do último estado consistente de checkpoints ser o estado inicial. Liu e Chen também propõem métricas para encontrar *checkpoints* úteis e descartar com segurança os *checkpoints* obsoletos ou inválidos. Desta forma não somente os *checkpoints* criados antes do conjunto consistente serão apagados, mas em caso de rollback, os que estão depois do conjunto consistente poderão ser eliminados, visto que não são mais úteis.

(MITTRA, 1995) propõe o uso do conceito de memória virtual no simulador *Verilog-XL*. O esquema proposto funciona de maneira parecida com os protocolos de memória cache. Se o endereço de memória necessário já está carregado na memória virtual, ele pode ser acessado diretamente, entretanto se ele não está carregado, é necessário fazer a carga antes de usá-lo. Esta proposta fornece a capacidade de ter, teoricamente, um espaço de memória infinito. No entanto, essa abordagem usa *swap* entre a memória principal e o armazenamento secundário, podendo deixar a simulação mais lenta.

Em (YOUNG; WILSEY, 1996) os autores citam uma implementação tradicional do *fossil collection*, onde o *timestamp* do item (*checkpoint*, log de mensagem, etc.) é comparado com o GVT. Assim somente as informações necessárias para fazer um rollback para o GVT são mantidas, e todas as informações anteriores ao GVT podem ser removidas. Os autores também apresentam uma nova técnica chamada *Optimistic Fossil Collection* (OFC) que estabelece modelos probabilísticos e usa um fator de risco definido pelo usuário para decidir se um item

pode ser recuperado. Basicamente, se a probabilidade for menor que o fator de risco o item pode ser eliminado. Assim como os próprios autores reconhecem, uma vez que a identificação é probabilística, há uma chance de ocorrerem erros.

Em (CAROTHERS; BAUER; PEARCE, 2000) é apresentado um novo mecanismo para simulações distribuídas chamado ROSS (*Rensselaer's Optimistic Simulation System*). Nesse mecanismo cada processador aloca um conjunto separado de memória para cada processador remoto, assim a memória é compartilhada apenas por um par de processadores. Quando a aplicação requisita memória, o processador de origem determina qual conjunto de memória do processador de destino será usado. Se o conjunto de memória livre no processador de destino estiver vazio o evento não é escalonado. Quando um evento termina sua execução, o cálculo do GVT é iniciado e protocolos de recuperação de memória retornam a memória livre para o conjunto de memória do processador proprietário. Uma vantagem dessa abordagem é que o excesso de otimismo é evitado, uma vez que a execução de processos em um processador é limitada pela sua memória livre.

Em (PARIZOTTO; MELLO, 2017) são propostas métricas para reduzir a média de *checkpoints* criados por meio da identificação de *checkpoints* inúteis. A estratégia é baseada em técnicas não coordenadas, não exigindo troca de mensagens exclusivamente para controle, mas sim usando o histórico de mensagens para decidir probabilisticamente se um *checkpoint* pode se tornar inútil. Esta solução foi implementada e analisada no DCB (*Distributed Simulation Backbone*) (MELLO, 2005).

4 METODOLOGIA

Atualmente não é comum o uso de mecanismos que limitam a memória utilizada para armazenar *checkpoints* criados em tempo de simulação. Modelos grandes ou que geram *checkpoints* com frequência podem levar a simulação a um *overflow* de memória e consequentemente a interrupção da simulação. Nesta seção são apresentadas as atividades previstas para o desenvolvimento de uma solução para gerenciamento de memória limitada para armazenamento de *checkpoints*.

O trabalho será dividido nas seguintes etapas: (*i*) Analisar técnicas já existentes de gerenciamento de memória limitada para armazenamento de *checkpoints*; (*ii*) Estudar políticas de substituição de *checkpoints* que possam ser incorporadas em um ambiente de simulação; (*iii*) Criar ou adaptar uma estratégia para controlar um espaço limitado de memória para armazenamento de *checkpoints*; (*iv*) Implementar a estratégia proposta no DCB; (*v*) Validar a estratégia criada com experimentos. (*vi*) Produzir um estudo para identificar com que tamanho, quantidade de *checkpoints* e frequência de criação de *checkpoints* uma simulação entrará em colapso por falta de espaço de armazenamento;

As etapas do trabalho serão divididas nas seguintes atividades:

Em (*i:i*) serão definidos critérios que podem estar relacionas ao gerenciamento de memória em um ambiente de simulação distribuída, para que em (*i:ii*) seja feita uma busca na bibliografia e em trabalhos relacionados. Em (*i:iii*) serão analisados os trabalhos encontrados para em (*i:iv*) identificar quão eficientes são essas soluções. Nesta etapa é esperado identificar os mecanismos com melhor custo-benefício, atendendo o primeiro objetivo específico.

Em (*ii:i*) serão buscadas políticas de substituição, para que em (*ii:ii*) sejam analisadas as características de cada política e em (*ii:iii*) seja feita uma classificação das políticas através de suas características. Nesta etapa, que contempla o segundo objetivo específico, é esperado encontrar políticas que possam ser adaptadas para um ambiente de simulação.

Em (*iii:i*) serão reunidas as informações obtidas em (*i*) e (*ii*), para que em (*iii:ii*) seja criada uma estratégia que controla um espaço limitado de armazenamento de *checkpoints*.

Em (*iv:i*) será feita uma análise do código do DCB para identificar onde é necessário implementar as funcionalidades de gerenciamento limitado de espaço de armazenamento de *checkpoints* e em (*iv:ii*) implementado o protocolo criado em (*iii*) na arquitetura do DCB. Em (*iv:iii*) serão realizados testes da implementação para verificar se as regras da estratégia desen-

volvida são respeitadas.

Em (*v:i*) serão criados modelos híbridos de simulação onde os *checkpoints* são criados baseados na troca de mensagens entre os componentes, permitindo que em (*v:ii*) seja feita uma comparação entre o protocolo criado em (*iii*) e o já existente. Assim será possível relacionar em (*v:iii*) o espaço de memória utilizado e mostrando em (*v:iv*) o impacto que a limitação da memória tem para o desempenho da simulação.

Com as etapas (*iii*), (*iv*) e (*v*) é esperado alcançar o terceiro objetivo específico.

Em (*vi:i*) será produzido um estudo em diversos tamanhos de simulações com diferentes frequências de criação de *checkpoints*, para em (*vi:ii*) identificar as condições em que uma simulação entrará em colapso por falta de memória, alcançando assim o último objetivo específico.

Atividade	Nov		Dez		Jan		Fev		Mar		Abr		Mai		Jun		Jul	
i.i	x																	
i.ii	x																	
i.iii		x																
i.iv		x																
ii.i			x															
ii.ii			x	x														
ii.iii				x														
iii.i					x													
iii.ii					x	x	x	x										
iv.i							x	x	x	x								
iv.ii									x	x	x	x						
iv.iii											x	x						
v.i													x					
v.ii													x	x				
v.iii													x	x				
v.iv															x			
vi.i															x	x	x	
vi.ii																	x	x

Tabela 4.1 – Cronograma de atividades

A tabela 4.1 mostra o cronograma das atividades e seus respectivos tempos de execução. A monografia será escrita durante todo o período de execução, detalhando a realização das atividades e os resultados obtidos.

4.1 Resultados Esperados

Desenvolvimento de um mecanismo de gerenciamento de memória que permita limitar o montante de memória utilizada pela criação de *checkpoints* sem interferir no desempenho e resultado da simulação. Isto deve eliminar o desperdício de armazenamento gerado pela criação de *checkpoints* e evitar o término da simulação por falta de memória.

REFERÊNCIAS

CAO, G.; SINGHAL, M. On Coordinated Checkpointing in Distributed Systems. **IEEE Trans. Parallel Distrib. Syst.**, Piscataway, NJ, USA, v.9, n.12, p.1213–1225, Dec. 1998.

CAROTHERS, C. D.; BAUER, D.; PEARCE, S. ROSS: a high-performance, low memory, modular time warp system. In: FOURTEENTH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2000. p.53–60. (PADS '00).

DAS, S. R.; FUJIMOTO, R. M. An Adaptive Memory Management Protocol for Time Warp Parallel Simulation. **SIGMETRICS Perform. Eval. Rev.**, New York, NY, USA, v.22, n.1, p.201–210, May 1994.

ELNOZAHY, E. N. M. et al. A Survey of Rollback-recovery Protocols in Message-passing Systems. **ACM Comput. Surv.**, New York, NY, USA, v.34, n.3, p.375–408, Sept. 2002.

FERSCHA, A.; TRIPATHI, S. K. **Parallel and Distributed Simulation of Discrete Event Systems**. College Park, MD, USA: [s.n.], 1994.

FUJIMOTO, R. M. Parallel and Distributed Simulation Systems. , [S.l.], 2000.

FUJIMOTO, R. M. Parallel and Distributed Simulation Systems. **Proceedings of the 2001 Winter Simulation Conference**, [S.l.], 2001.

KUMAR, S.; CHAUHAN, R. K.; KUMAR, P. Real Time Snapshot Collection Algorithm for Mobile Distributed Systems with Minimum Number of Checkpoints. , [S.l.], 2010.

LAW, A. M.; KELTON, W. D. **Simulation Modeling and Analysis**. 2nd.ed. [S.l.]: McGraw-Hill Higher Education, 1997.

LIU, Y.; CHEN, J. Garbage collection in uncoordinated checkpointing algorithms. **Journal of Computer Science and Technology**, [S.l.], v.14, p.242–249, 1999.

MELLO, B. A. d. Co-Simulação Distribuída de Sistemas Heterogêneos. **Tese (Doutorado em Ciência da Computação)**, [S.l.], 2005.

MITTRA, S. A Virtual Memory Management Scheme for Simulation Environment. In: IEEE INTERNATIONAL VERILOG HDL CONFERENCE, 4., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 1995. p.114–. (IVC '95).

PARIZOTTO, R.; MELLO, B. A. d. Heuristic checkpointing on a parallel heterogeneous discrete event simulation system. , [S.l.], 2017.

PIDD, M. An Introduction to Computer Simulation. In: CONFERENCE ON WINTER SIMULATION, 26., San Diego, CA, USA. **Proceedings...** Society for Computer Simulation International, 1994. p.7–14. (WSC '94).

PREISS, B. R.; MACINTYRE, I. D.; LOUCKS, W. M. On the Trade-off between Time and Space in Optimistic Parallel Discrete-Event Simulation. , [S.l.], 1992.

QUAGLIA, F. Combining Periodic and Probabilistic Checkpointing in Optimistic Simulation. In: THIRTEENTH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 1999. p.109–116. (PADS '99).

SATO, K. et al. Design and Modeling of a Non-blocking Checkpointing System. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS, Los Alamitos, CA, USA. **Proceedings...** IEEE Computer Society Press, 2012. p.19:1–19:10. (SC '12).

SHANNON, R. E. Introduction to the Art and Science of Simulation. In: CONFERENCE ON WINTER SIMULATION, 30., Los Alamitos, CA, USA. **Proceedings...** IEEE Computer Society Press, 1998. p.7–14. (WSC '98).

WANG, Y.-M. **Space Reclamation for Uncoordinated Checkpointing in Message-passing Systems**. 1993. Tese (Doutorado em Ciência da Computação) — , Champaign, IL, USA. UMI Order No. GAX94-11816.

YOUNG, C. H.; WILSEY, P. A. Optimistic Fossil Collection for Time Warp Simulation. **Proceedings of the 29th Annual Hawaii International Conference on System Sciences**, [S.l.], 1996.