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

EDUARDO OGLIARI

PROPOSTA DE MÉTRICAS PARA O GARBAGE COLLECTOR ASSÍNCRONO RPGC

CHAPECÓ
2022

EDUARDO OGLIARI

PROPOSTA DE MÉTRICAS PARA O GARBAGE COLLECTOR ASSÍNCRONO RPGC

Trabalho de conclusão de curso 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
2022

Bibliotecas da Universidade Federal da Fronteira Sul - UFFS

Ogliari, Eduardo
Proposta de métricas para o garbage collector assíncrono RPGC / Eduardo Ogliari. -- 2022.
47 f.:il.

Orientador: Prof. Dr. Braulio Adriano de Mello

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal da Fronteira Sul, Curso de Bacharelado em Ciência da Computação, Chapecó,SC, 2022.

1. Simulação Distribuída. 2. Garbage Collection Assíncrono. 3. RPGC. I. Mello, Braulio Adriano de, orient. II. Universidade Federal da Fronteira Sul. III. Título.

Elaborada pelo sistema de Geração Automática de Ficha de Identificação da Obra pela UFFS com os dados fornecidos pelo(a) autor(a).

EDUARDO OGLIARI

PROPOSTA DE MÉTRICAS PARA O GARBAGE COLLECTOR ASSÍNCRONO RPGC

Trabalho de conclusão de curso 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 avaliadora em: 19/8/2022.

BANCA AVALIADORA

Prof. Dr. Braulio Adriano de Mello – UFFS

Prof. Me. Adriano Sanick Padilha – UFFS

Prof. Dr. Marco Aurélio Spohn – UFFS

RESUMO

Simulações distribuídas necessitam de estratégias de sincronização de tempo para evitar que eventos de simulação sejam executados fora da ordem de tempo. Estratégias conservadoras impedem essa violação de tempo. Estratégias otimistas permitem que violações de tempo ocorram, porém, retrocedem a simulação a um estado seguro anterior denominado *checkpoint*. Com o avanço do tempo de simulação, a quantidade de *checkpoints* salvos tende a aumentar indefinidamente. Este cenário pode resultar no esgotamento da capacidade de armazenamento de *checkpoints* e, então, colapso da simulação. Estratégias de sincronização otimistas são classificadas em coordenadas, não-coordenadas e orientadas a comunicação. Na sincronização coordenada, remover *checkpoints* é um processo trivial. Na sincronização não-coordenada e orientada à comunicação existem algoritmos de *garbage collection* que são executados periodicamente para atuar na remoção de *checkpoints*. O *Rollback Prediction Garbage Colletor* (RPGC) é um *garbage collector* assíncrono que faz uso de métricas para determinar quantos *checkpoints* devem ser mantidos. Entretanto, as métricas atuais do RPGC tendem a manter *checkpoints* que não serão mais utilizados em operações de *rollback* futuras. Este trabalho apresenta a implementação de uma nova métrica e um novo parâmetro; a frequência de *rollbacks* e a porcentagem de crescimento do maior *rollback*, respectivamente. Através destas, resultados satisfatórios em melhorar a precisão das estimativas foram obtidos. A implementação foi integrada na arquitetura de simulação distribuída DCB (*Distributed Co-Simulation Backbone*).

Palavras-chave: Simulação Distribuída. Garbage Collection Assíncrono. RPGC.

ABSTRACT

Distributed simulations require time synchronization strategies to prevent simulation events from being processed out of timestamp order. Conservative strategies prevent time violation. Optimistic strategies allow time violations to happen but rollback the simulation to a previously safe state, denominated checkpoint. As the simulation advances, the amount of checkpoints tend to grow, which results in memory usage increase. Optimistic synchronization strategies are classified in coordinated, uncoordinated e communication induced. In coordinated synchronization, removing checkpoints is a trivial task. In uncoordinated and communication-induced synchronization strategies there are garbage collection algorithms that are executed periodically in the removal of checkpoints. RPGC is an asynchronous garbage collector that makes use of metrics to determine the amount of checkpoints that should remain in the simulation. However, the existing metrics used tend to keep checkpoints that are not going to be used in future rollback operations. We present the implementation of one new metric and one new parameter – rollback frequency and the growth percentage of the largest rollback, respectively. Satisfactoy results were obtained in improving the precision of the estimates. Furthermore, some fragilities were identified in the original work. The implementation was integrated into the Distributed Co-Simulation Backbone, a distributed simulation architecture.

Keywords: Distributed Simulation. Asynchronous Garbage Collection. RPGC.

LISTA DE ILUSTRAÇÕES

Figura 1 – Violação de LCC 20
Figura 2 – Multiplicador de segurança ao longo da simulação (1) 23
Figura 3 – Arquitetura do DCB (6) 25
Figura 4 – *Checkpoints* armazenados em um cenário com $S_{min} = 1$ 28
Figura 5 – Progressão de S_{mult} com $S_{min} = 1$ 29
Figura 6 – Modelo utilizado nos estudos de caso 33
Figura 7 – Avanço de *lvt* (esquerda: $\sigma = 10$, direita: $\sigma = 100$) 34
Figura 8 – Evolução de *checkpoints* para teste T_5 (esquerda: trabalho original, direita: este trabalho) 36
Figura 9 – Evolução de S_{mult} para teste T_5 (esquerda: trabalho original, direita: este trabalho) 37
Figura 10 – Influência das métricas e parâmetros combinados para o teste T_5 37
Figura 11 – Evolução de *checkpoints* para teste T_1 (esquerda: trabalho original, direita: este trabalho) 38
Figura 12 – Evolução de S_{mult} para teste T_1 (esquerda: trabalho original, direita: este trabalho) 38
Figura 13 – Influência das métricas e parâmetros combinados para o teste T_1 39
Figura 14 – Progresso de F_{roll} para o teste T_1 40
Figura 15 – Diferença entre $R_{max} \times P_{avg}$ e R_{max} para o teste T_1 41
Figura 16 – Progresso de P_{avg} para o teste T_1 41
Figura 17 – Progressão na quantidade de *checkpoints* após a ocorrência de um *rollback* desproporcional 42
Figura 18 – Progressão da variabilidade em uma simulação que encerrou prematuramente 43

LISTA DE ALGORITMOS

Algoritmo 1 – Calculo de P_{avg} executado apos um *rollback* 31
Algoritmo 2 – Cálculo do RPGC de quantos *checkpoints* devem ser mantidos com base nas métricas coletadas 31

LISTA DE TABELAS

Tabela 1 – Constantes de configuração do RPGC 24
Tabela 2 – Métricas do RPGC 24
Tabela 3 – Parâmetros do RPGC 24
Tabela 4 – Distribuição utilizada em cada teste 34
Tabela 5 – Comparativo da quantidade de *checkpoints* para cada teste realizado 35
Tabela 6 – Comparativo do valor médio de F_{roll} para cada teste realizado 40
Tabela 7 – Comparativo do valor de P_{avg} para cada teste realizado 40

SUMÁRIO

1	**INTRODUÇÃO E CONTEXTUALIZAÇÃO DO PROBLEMA**	**17**
2	**REFERENCIAL TEÓRICO**	**19**
2.1	SIMULAÇÃO	19
2.2	SIMULAÇÃO DISTRIBUÍDA	19
2.3	SINCRONIZAÇÃO	19
2.3.1	**Sincronização Conservadora**	**20**
2.3.2	**Sincronização Otimista**	**20**
2.4	EFEITO CASCATA	20
2.5	ROLLBACK BASEADO EM CHECKPOINTS	21
2.6	ROLLBACK-DEPENDENCY TRACKABILITY	21
2.7	GARBAGE COLLECTION	22
2.8	ROLLBACK PREDICTION GARBAGE COLLECTOR (RPGC)	22
2.8.1	**Etapas de execução**	**22**
2.8.2	**Métricas e parâmetros**	**23**
2.9	DISTRIBUTED CO-SIMULATED BACKBONE	24
2.10	TRABALHOS CORRELATOS	25
3	**ESPECIFICAÇÃO DA SOLUÇÃO**	**27**
3.1	ANÁLISE DAS MÉTRICAS ATUAIS	27
3.2	SOLUÇÃO PROPOSTA	29
3.3	IMPLEMENTAÇÃO	30
4	**EXPERIMENTAÇÃO, RESULTADOS E ANÁLISES**	**33**
4.1	ESTUDOS DE CASO	33
4.2	RESULTADOS E ANÁLISES	35
4.2.1	**Quantidade média de *checkpoints***	**35**
4.2.2	**Frequência de *rollbacks***	**39**
4.2.3	**Porcentagem média de crescimento do maior *rollback***	**40**
4.2.4	**Variabilidade**	**42**
5	**CONCLUSÃO**	**45**
5.1	TRABALHOS FUTUROS	45
	REFERÊNCIAS	**47**

1 INTRODUÇÃO E CONTEXTUALIZAÇÃO DO PROBLEMA

Simulações computacionais são amplamente utilizadas para analisar o comportamento de sistemas reais ou imaginários em relação ao tempo. Através da representação do comportamento é possível inferir como um sistema existente ou hipotético pode se comportar sob diversas situações, sobretudo em sistemas reais que podem possuir riscos e custos elevados.

Uma estratégia adotada é a divisão de uma simulação em partes que possam ser executadas em nodos distintos. A distribuição de partes de um modelo traz benefícios como por exemplo, uma melhora no desempenho geral da simulação. A tal estratégia incide em problemas de sincronização de eventos de simulação que requerem controles específicos para garantir a correção dos resultados de simulação. Este trabalho tem como tema geral a sincronização em simulação distribuída, cujo problema alvo específico é detalhado a seguir.

Simulações podem ter sua execução distribuída através de múltiplos computadores diferentes. Algumas possíveis vantagens dessa abordagem incluem um tempo reduzido de execução, a integração de simuladores de fabricantes distintos à uma mesma simulação, e uma tolerância maior à falhas. Entretanto, devido a execução da simulação ocorrer de forma distribuída, existe a possibilidade de eventos de simulação serem executados fora de ordem, o que afeta a a coerência do resultado produzido pela simulação (3).

Algoritmos de sincronização de tempo são utilizados para que os eventos de uma simulação distribuída não sejam executados fora de ordem, seja por meio de estratégias de prevenção ou de recuperação (3). Algoritmos que utilizam estratégias de recuperação adotam operações de salvamento de *checkpoints* e operações de *rollback*, que permitem a simulação retroceder à um estado seguro em uma eventual violação de tempo.

Mecanismos de *rollback* que fazem uso de *checkpoints* podem ser classificados em coordenados, não-coordenados e orientados à comunicação. Tanto os mecanismos não-coordenados como os orientados à comunicação estão sujeitos ao efeito cascata, que ocorre quando *rollbacks* sucessivos acabam retrocedendo a simulação até o inicio da computação (2). No entanto, mecanismos de *rollback* orientados a comunicação conseguem evitar o efeito cascata ao identificarem as dependências entre os componentes da simulação (13).

Com o avanço do tempo de simulação, o uso de memória tende a aumentar de maneira expressiva devido à criação de novos *checkpoints*, o que pode levar a simulação a um fim prematuro. Trabalhos foram realizados para tratar da eliminação de *checkpoints* que não serão mais úteis em operações de *rollback* futuras em uma simulação. De modo geral, esses trabalhos consistem em determinar uma linha de recuperação segura mais recente, e remover todos os *checkpoints* que a precedem (2).

Soluções coordenadas e induzidas a comunicação tendem a possuir um uso maior de recursos em relação à soluções não-coordenadas devido a necessidade de coordenação global. No entanto, soluções não-coordenadas encontram dificuldades em identificar precisamente *checkpoints* úteis e inúteis. Assim, soluções não-coordenadas precisam armazenar uma quantidade

maior de *checkpoints* para evitar a remoção de *checkpoints* que ainda podem ser utilizados em *rollbacks* futuros (2).

O *Rollback Predictor Garbage Collector* (RPGC) (1) é um *garbage collector* assíncrono que utiliza métricas para estimar uma quantidade segura de *checkpoints* a serem mantidos na simulação. Por ser um *garbage collector* assíncrono, trata-se de uma solução que independe de coordenação. Pelo fato do *RPGC* trabalhar com estimativas, ainda é possível que *checkpoints* úteis sejam removidos causando o colapso da simulação, e que *checkpoints* não-úteis não sejam removidos causando desperdício de memória.

Visando melhorar a precisão da estimativa da quantidade de *checkpoints* do *RPGC*, as contribuições deste trabalho consistiram da integração de uma nova métrica e um novo parâmetro; a frequência de *rollbacks* e a a porcentagem de crescimento do maior *rollback*, respectivamente. Os testes executados mostraram que as mudanças realizadas foram capazes de tornar o cálculo da estimativa de *checkpoints* mais preciso. A solução foi implementada no *Distributed Co-Simulation Backbone* (DCB), uma arquitetura de simulação distribuída.

No Capítulo 2 é realizado uma revisão bibliográfica acerca dos tópicos abordados neste trabalho, assim como são apresentados trabalhos correlatos à este. No capítulo 3 são detalhadas as métricas utilizadas neste trabalho, assim como sua implementação. Por fim, o capítulo 4 apresenta os resultados obtidos pelas novas métricas e o capítulo 5 as considerações finais.

2 REFERENCIAL TEÓRICO

2.1 SIMULAÇÃO

Uma simulação é um sistema que modela o comportamento de um outro sistema real ou imaginário em relação ao tempo (3). Simulações são utilizadas principalmente para analisar o comportamento de sistemas que podem ter custo e riscos elevados no mundo real.

Simulações são geralmente divididas em dois grupos: discretas e contínuas (8). Em uma simulação discreta o estado da simulação muda através de eventos no tempo, e o progresso ocorre de maneira sequencial, de evento para evento. Em uma simulação contínua as mudanças de estado ocorrem conforme o avanço do tempo.

2.2 SIMULAÇÃO DISTRIBUÍDA

Uma simulação pode ter sua execução distribuída em vários componentes cooperantes que podem estar geograficamente dispersos. Alguns benefícios desta estratégia incluem um tempo de execução reduzido, maior tolerância à falhas e a integração de simuladores de fabricantes diferentes (3). Entretanto, esta estratégia exige a sincronização entre os componentes de uma simulação, pois caso contrário os eventos trocados entre os componentes podem ser processados fora de ordem, causando inconsistências na simulação.

2.3 SINCRONIZAÇÃO

Simulações de eventos discretos necessitam que os eventos trocados sejam processados em ordem de tempo de execução. As mensagens trocadas entre os componentes de uma simulação contém informações de eventos, sendo uma delas o tempo em que um evento deve ser processado, denominado *timestamp*. Cada componente da simulação possuí um valor de *Local Virtual Time* (LVT) armazenado localmente. Esse valor corresponde ao *timestamp* do último evento executado pelo componente (3).

O *Local Causality Constraint* (LCC) é uma propriedade que afirma que todos os processos lógicos de uma simulação distribuída que se comunicam por meio de troca de mensagens devem processar os eventos respeitando a ordem de *timestamp* (3). Uma violação de LCC ocorre quando um evento foi processado fora da ordem de *timestamp*, causando inconsistências e afetando o resultado produzido pela simulação.

Na figura 1 temos um exemplo de violação de LCC. O processo P_1 e P_2 possuem valores de LVT t_1 e t_3, respectivamente. Durante a execução de um evento por P_1 no instante t_1, um evento é gerado e uma mensagem enviada para P_2 para ser processada no instante t_2. Entretanto, o processo P_2 possuí um valor de LVT superior ao instante em que a mensagem deveria ser processada, causando assim uma violação de tempo.

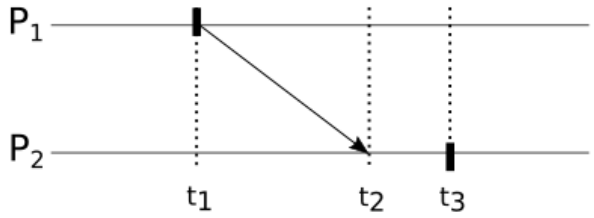

Figura 1 – Violação de LCC

2.3.1 Sincronização Conservadora

Algoritmos de sincronização conservadora tentam impedir que processos lógicos executem eventos fora da ordem correta de *timestamp*. Além disso, estão sujeitos a situações de *deadlock* que devem ser evitadas (3).

2.3.2 Sincronização Otimista

Algoritmos de sincronização otimista permitem que violações de tempo ocorram, porém retrocedem a simulação para um estado seguro anterior ao instante de tempo em que ocorreu a violação de tempo. Um mecanismo de controle global denominado *Global Virtual Time* (GVT) é utilizado para determinar um *snapshot* global instantâneo da simulação. O GVT corresponde ao menor valor de LVT entre todos os processos lógicos da simulação (4)

O cálculo de GVT pode ser realizado de maneira síncrona ou assíncrona. Um algoritmo síncrono pode ser implementado por meio da troca de mensagens entre um controlador central e os componentes lógicos da simulação distribuída. No entanto, isso exige o bloqueio do processamento de todos os componentes lógicos (3). Mattern propôs um algoritmo de cálculo de GVT assíncrono que não requer troca de mensagens (5). Cada processo lógico define um ponto de corte em sua execução que divide a computação em passado e presente. O conjunto de todos os pontos de corte define um corte consistente, onde violações de LCC não ocorrem. Assim, o valor de GVT pode ser calculado com base no corte consistente da simulação.

O problema-alvo deste trabalho está relacionado à sincronização otimista, cujos tópicos a seguir estão diretamente associados.

2.4 EFEITO CASCATA

Quando um *rollback* é realizado, todos os eventos que aconteceram depois do *checkpoint* para o qual o *rollback* será realizado serão cancelados. No entanto, caso as mensagens geradas pelos eventos cancelados já tenham sido processadas pelos processos lógicos de destino, estas mensagens se tornam órfãs. Quaisquer processos lógicos que processaram mensagens órfãs

devem realizar um *rollback* para um *checkpoint* de *timestamp* anterior ao *timestamp* da mensagem órfã. Essa situação pode resultar em um efeito cascata, que ocorre quando um *rollback* acaba desencadeando uma série de *rollbacks* que retrocede a computação para o início da simulação (3).

2.5 ROLLBACK BASEADO EM CHECKPOINTS

Abordagens que fazem uso de *checkpoints* em operações de *rollback* restauram a simulação para o conjunto consistente mais recente de *checkpoints*, denominado linha de recuperação. Protocolos de tomada de *checkpoint* são classificados em não-coordenados, coordenados e induzidos por comunicação (2).

A tomada de *checkpoints* de maneira não-coordenada permite que cada componente lógico decida o momento mais apropriado para criar um *checkpoint*. No entanto, pode acabar criando *checkpoints* que nunca serão parte de um estado global consistente, assim como também está sujeito ao efeito cascata. Por outro lado, a tomada de *checkpoints* de maneira coordenada requer que todos os processos lógicos troquem mensagens para coordenar a criação de *checkpoints*. O efeito cascata é evitado pois em caso de *rollback* cada processo recomeça do seu último checkpoint mais recente. Em protocolos de criação de *checkpoints* induzidos por comunicação, os processos lógicos criam seus *checkpoints* de maneira independente. Diferente de protocolos coordenados, protocolos induzidos por comunicação não fazem uso de mensagens de coordenação – ao invés disso, utilizam informações adicionais contidas em mensagens de aplicação (2).

2.6 ROLLBACK-DEPENDENCY TRACKABILITY

Um *checkpoint* local representa o estado de um processo lógico. Considere que $C_{i,x}$ representa o *x-ésimo checkpoint* local do processo i, e que $\mathrm{I}_{i,x}$ denota o intervalo de *checkpoint* entre $C_{i,x-1}$ e $C_{i,x}$. Dizemos que um *checkpoint* $C_{i,x}$ depende diretamente de outro *checkpoint* $C_{j,y}$ se $i \neq j$ e uma mensagem m é enviada de $\mathrm{I}_{i,x}$ e recebida em $\mathrm{I}_{j,y}$, ou se $i = j$ e $y = x + 1$ (13). Assim, a remoção do *checkpoint* $C_{i,x}$ implica na remoção do *checkpoint* $C_{j,y}$.

O *Rollback-Dependency Trackability* (RDT) é uma propriedade que permite cada processo lógico determinar suas dependências por meio da propagação de um vetor transitivo de dependências.

Assuma que existam N processos lógicos em uma simulação. Cada processo lógico P_i i possui um vetor D_i de tamanho N. A entrada correspondente ao próprio processo lógico $\mathrm{D}_i[i]$ é inicializada em 1 e incrementada toda vez que um novo *checkpoint* é criado. As demais entradas $\mathrm{D}_i[j]$ são inicializadas em 0 e armazenam os maiores índices de *checkpoints* de P_j no qual o estado atual de P_i depende transitivamente. Quando uma mensagem m é enviada de P_i para P_j,

o vetor D_i é propagado junto com m. O processo P_j atualiza seu vetor D_j com o valor máximo entre os componentes D_j e D_i, ou $D_j[k] = max(D_j[k], D_i[k])$ (13).

2.7 GARBAGE COLLECTION

Com o avanço da simulação, a quantidade de *checkpoints* criados tende a crescer. Cada *checkpoint* armazenado consome uma quantidade significativa de memória. *Checkpoints* antigos tendem a não ser mais utilizados em operações de recuperação conforme novos *checkpoints* são criados. O processo de remoção de *checkpoints* que não são mais considerados úteis é denominado *garbage collection*. Uma estratégia simples consiste da identificação da linha de recuperação mais recente da simulação, e da remoção de todos os *checkpoints* que precedem essa linha (11).

Um algoritmo de *garbage collection* é considerado assíncrono se e somente se ele depende apenas de informações propagadas por mensagens de aplicação (12).

2.8 ROLLBACK PREDICTION GARBAGE COLLECTOR (RPGC)

O RPGC é um *garbage collector* assíncrono que utiliza métricas para calcular uma estimativa segura de *checkpoints* mais recentes a serem mantidos na simulação. No contexto do RPGC, o tamanho de um *rollback* é definido como a quantidade de *checkpoints* retrocedidos por esse mesmo *rollback*. As métricas foram estabelecidas partindo do pressuposto que padrões de *rollback* tendem a se repetir no futuro. Além disso, constantes de configuração podem ser modificadas para alterar o comportamento do algoritmo.

2.8.1 Etapas de execução

A execução do RPGC acontece em três etapas. Na primeira etapa o algoritmo não remove nenhum *checkpoint*, e se limita apenas a atualizar os valores das métricas. Isso ocorre pois no início da simulação ainda há pouca informação para realizar as previsões com segurança, o que tornaria as estimativas imprecisas e frágeis. Assim, a primeira etapa é encerrada apenas a partir do momento que o componente receba uma quantidade de mensagens excedente ao valor de T_{wait} mensagens.

A partir da segunda etapa a remoção de *checkpoints* é permitida. O multiplicador de segurança S_{mult} é utilizado para aumentar a quantidade de *checkpoints* que permanecem na simulação. Inicialmente S_{mult} assume o valor de S_{init}, e vai assumindo valores menores a cada múltiplo de S_{dec} mensagens recebidas até estabilizar no valor de S_{min}. Na terceira etapa, S_{mult} assume o valor de S_{min} e não sofre mais alterações. Essa estratégia é utilizada para dar mais volume as estimativas iniciais por ainda não serem tão precisas. Esse comportamento é ilustrado na figura 2

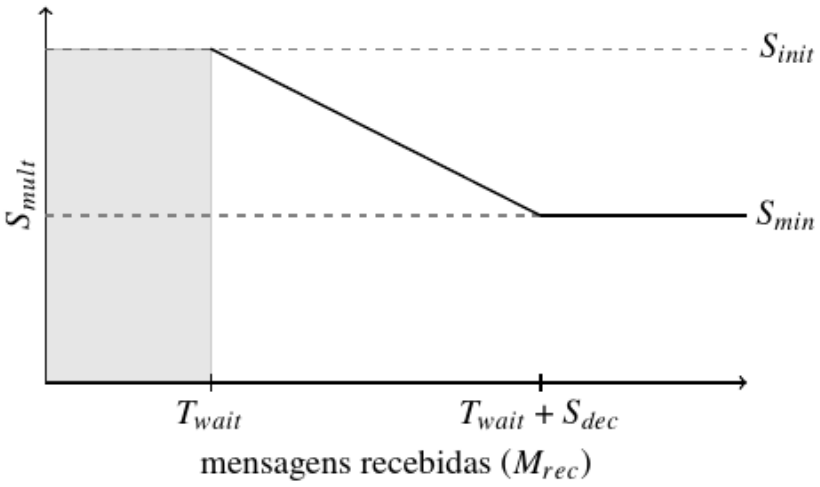

Figura 2 – Multiplicador de segurança ao longo da simulação (1)

Todos os *checkpoints* salvos pela simulação são armazenados em uma lista dinâmica, e todo novo *checkpoint* salvo é inserido no final dessa lista. No processo de remoção de *checkpoints*, os C_{keep} itens do final da lista permanecem na memória, e os demais itens são removidos. Em outras palavras, se a lista de *checkpoints* possuí tamanho n, então o intervalo que representa os *checkpoints* que permanecem na simulação é $[n - Ckeep, n]$, enquanto os demais *checkpoints* pertencentes ao intervalo $[0, n - Ckeep)$ são removidos.

2.8.2 Métricas e parâmetros

Toda vez que um *rollback* é realizado, os valores de R_{max} e R_{avg} são atualizados. Seja R_{len} o tamanho de um *rollback* que acabou de ser realizado, então o valor de R_{max} é calculado através da função $R_{max} = max(R_{max}, R_{len})$. A média de *rollbacks* R_{avg} é calculada como uma média móvel exponencial para eliminar a necessidade de armazenar o tamanho de todos os *rollbacks* em uma lista (1). Seja R^n_{avg} a média de *rollbacks* após ocorrerem n *rollbacks*, então:

$$R^{n+1}_{avg} = \begin{cases} n = 0, & R_{len} \\ n \geq 1, & R^n_{avg} \times (1 - W) + R_{len} \times W \end{cases}$$

A variabilidade de *rollbacks* V é calculada como a diferença entre o tamanho do maior *rollback* R_{max} e a média de tamanhos de *rollback* R_{avg}. Esse valor é utilizado para tentar prever o quão grande pode ser um *rollback* futuro em relação ao maior *rollback* até então registrado (1). A variabilidade mínima V_{min} é um valor definido pelo usuário, e é utilizada para garantir que a variabilidade não assuma valores muito pequenos.

A margem de segurança S é resultante da multiplicação entre a variabilidade V e o multiplicador de segurança S_{mult}. Por fim, esse valor é somado ao tamanho do maior *rollback* R_{max} para determinar a quantidade de *checkpoints* C_{keep} que devem permanecer na simulação.

É importante ressaltar que o valor de S deve ser calculado de modo que não seja baixo demais e nem alto demais. Um valor baixo implica uma margem de segurança que leva à estimativas frágeis demais, correndo o risco de eliminar *checkpoints* que ainda podem ser utilizados em operações de *rollback*. Em contrapartida, um valor alto de margem de segurança leva à uma quantidade elevada de *checkpoints* não-úteis mantidos na simulação, aumentado o desperdício de memória.

A tabela 1 exibe as constantes de configuração, a tabela 2 exibe a listagem as métricas, e a tabela 3 a listagem dos parâmetros.

V_{min}	Variabilidade mínima
S_{dec}	Decremento do multiplicador de segurança
S_{min}	Valor mínimo do multiplicador de segurança
T_{wait}	Período de espera (em mensagens recebidas)
C_{min}	Quantidade mínima de *checkpoints* a serem mantidos
W	Peso dado ao *rollback* mais recente para cálculo da média móvel exponencial de *rollbacks*

Tabela 1 – Constantes de configuração do RPGC

R_{max}	Tamanho do maior *rollback*
R_{avg}	Média de tamanhos de *rollback*
M_{rec}	Contador de mensagens recebidas

Tabela 2 – Métricas do RPGC

$V = max(V_{min}, R_{max} - \lfloor R_{avg} \rfloor)$	Variabilidade
$S_{mult} = max(S_{init} - \frac{M_{rec} - T_{wait}}{S_{dec}}, S_{min})$	Multiplicador de segurança
$S = \lfloor V \times S_{mult} \rfloor$	Margem de segurança
$C_{keep} = max(C_{min}, R_{max} + S)$	Quantidade de *checkpoints* a serem mantidos

Tabela 3 – Parâmetros do RPGC

2.9 DISTRIBUTED CO-SIMULATED BACKBONE

O *Distributed Co-Simulated Backbone* (DCB) é uma arquitetura de suporte à simulação distribuída de modelos heterogêneos. Sua flexibilidade permite a interoperabilidade entre componentes com características distintas (6).

Conforme ilustrado na figura 3, a arquitetura do DCB é composta de quatro módulos principais. O Embaixador de Federado (EF) gerencia mensagens recebidas de outros federados. O Embaixador do DCB (EDCB) gerencia mensagens emitidas pelo federado que representa e gerencia o LVT em conjunto com o EF. O Núcleo do DCB gerencia os serviços de troca de mensagens e armazena o valor de GVT. Por fim, o *gateway* age como uma interface entre o federado e os demais módulos do DCB (6).

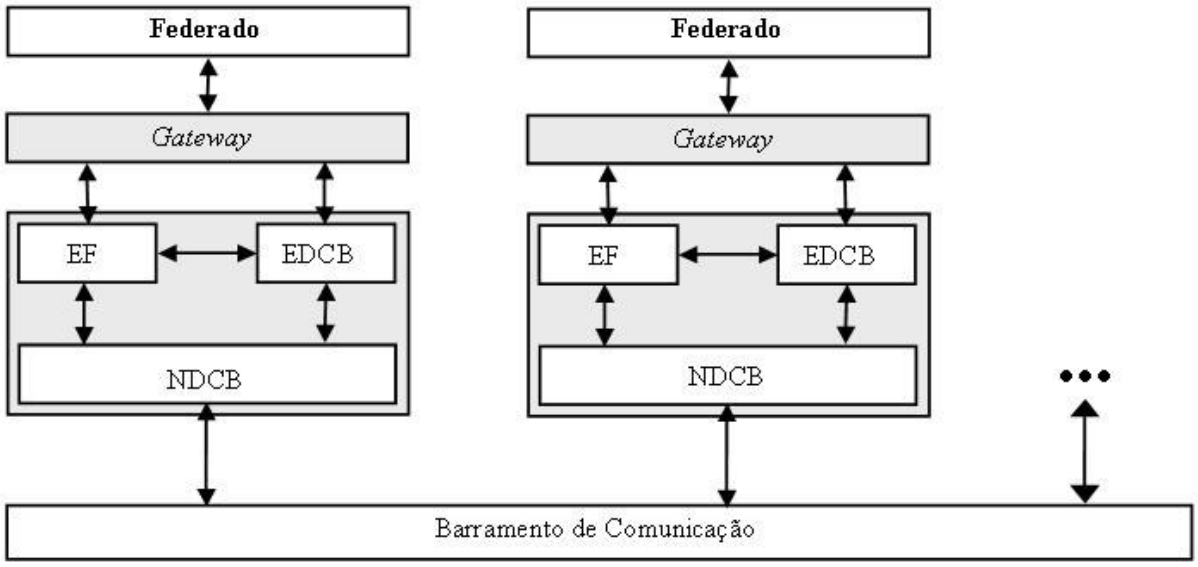

Figura 3 – Arquitetura do DCB (6)

2.10 TRABALHOS CORRELATOS

Nesta seção são apresentados trabalhos relacionados ao objetivo deste trabalho.

O trabalho de (ADAMCZUK, 2021) (1) apresenta o RPGC, um algoritmo de *garbage collection* assíncrono sub-ótimo que utiliza métricas para estimar quantos *checkpoints* devem permanecer na simulação. A quantidade de *checkpoints* mantidos é calculada com base no tamanho do maior *rollback* registrado, acrescido ao valor de uma margem de segurança.

O conceito de *Rollback-Dependency Trackability* é apresentado em (WANG, 1997)(13). Essa propriedade permite identificar a dependência entre *checkpoints* de uma simulação utilizando um vetor de dependências transitivo. Assim, é possível realizar o cálculo da linha de recuperação através de soluções descentralizadas.

O algoritmo de *garbage collection* RDT-LGC foi introduzido no trabalho de (SCHMIDT, 2005) (12). Trata-se de um *garbage collector* voltado para protocolos de *checkpoint* induzidos à comunicação que garantem a propriedade RDT. Dessa forma, o algoritmo não requer o uso de sincronização explícita entre processos, apenas da propagação de informação através das mensagens de aplicação.

Em (QUAGLIA, 2001)(10), é apresentado uma estratégia de criação de *checkpoints* por meio de um modelo de custo que determina a conveniência de criar um *checkpoint* no estado atual. Um *checkpoint* é criado caso o custo de salvar um *checkpoint* seja inferior ao de não salvar. Neste trabalho foi introduzido o cálculo da frequência de *rollbacks*, que é utilizado para determinar a probabilidade de um *rollback* acontecer dentro de um intervalo de tempo específico.

No trabalho de (PARIZOTTO; MELLO, 2019)(7) foi apresentado uma estratégia para a identificação de *checkpoints* não-úteis através do uso de métricas. O método utiliza padrões de comunicação e a granularidade dos eventos desde o último *rollback*. A estratégia permite minimizar a quantidade de *checkpoints* não-úteis sem impactar negativamente a performance da simulação.

Em (PUTTLITZ, 2021) (9) foi apresentado um mecanismo para adaptar os intervalos entre *checkpoints* se baseando em um modelo de custo de recuperação para um determinado estado da simulação em situações de *rollback*. A frequência de *rollbacks* foi utilizada para determinar a probabilidade de um estado ser restaurado futuramente em caso de *rollback*.

3 ESPECIFICAÇÃO DA SOLUÇÃO

Este capítulo inicia com uma análise das métricas atuais do RPGC, detalhando o funcionamento e eficiência do algoritmo, assim como as fragilidades observadas. Em seguida são apresentadas as novas métricas e a motivação pelas escolhas. Por fim é apresentado uma implementação em pseudocódigo das alterações realizadas no algoritmo.

3.1 ANÁLISE DAS MÉTRICAS ATUAIS

O objetivo do RPGC é o de utilizar métricas para gerar estimativas de o quão grande os *rollbacks* podem se tornar futuramente. Através dessas estimativas, é possível armazenar uma quantidade suficiente de *checkpoints* na memória, e eliminar o excesso. Caso a estimativa seja pequena demais, é possível que faltem *checkpoints* para as operações de *rollback*. Se a estimativa for grande demais, haverá desperdício de memória.

Buscando manter uma quantidade suficiente de *checkpoints* na memória, o RPGC gera uma estimativa através da métrica de tamanho do maior *rollback* registrado R_{max} e o parâmetro de margem de segurança S. O tamanho de um *rollback* é dado pela quantidade de *checkpoints* retrocedidos por esse mesmo *rollback*.

A métrica R_{max} tende a aumentar conforme os *rollbacks* vão ficando maiores. Assim, ao menos R_{max} *checkpoints* são estimados a serem mantidos na memória durante a execução, pois existe a possibilidade de um *rollback* do mesmo tamanho ocorrer novamente. Essa é a estimativa base do RPGC.

A margem de segurança S busca complementar a estimativa base R_{max}, partindo do princípio que *rollbacks* ainda maiores possam ocorrer no futuro. Para realizar essa estimativa complementar, é levado em consideração o quanto os *rollbacks* estão variando em tamanho. Isso é obtido por meio do cálculo da diferença entre R_{max} e a média de tamanho de *rollbacks* R_{avg}, denominado a variabilidade de *rollbacks* V da simulação. Caso V assuma um valor baixo, significa que os *rollbacks* estão assumindo tamanhos bem próximos, com pouca variação de tamanho. Um valor alto de V indica que os *rollbacks* estão variando mais em tamanho. Assim, quanto mais os *rollbacks* estão variando em tamanho, maior a quantidade de *checkpoints* que devem ser mantidos, pois há menos garantias em relação ao tamanho que os *rollbacks* podem assumir.

No entanto, a variabilidade pode assumir valores pequenos e insuficientes para as previsões. O multiplicador de segurança S_{mult} desempenha o papel de ampliar a variabilidade V nos estágios iniciais de execução para compor o valor final da margem de segurança S. Quanto maior o valor de S_{mult}, maior a compensação aplicada a variabilidade. Como se trata de um valor dinâmico, S_{mult} inicia com o valor da constante S_{init}, e vai assumindo valores menores até estabilizar no valor definido pela constante S_{min}. O decréscimo de S_{mult} aumenta proporcionalmente a quantidade de mensagens recebidas, em múltiplos do valor de S_{dec} mensagens.

Por fim, para gerar a estimativa final de C_{keep} *checkpoints* a serem mantidos na memória, o valor da margem de segurança S é então somado ao tamanho do maior *rollback* R_{max}.

Entretanto, foi observado que as estimativas geradas pelas métricas atuais do RPGC são dependentes de um valor de $S_{min} > 1$. Caso contrário, as estimativas acabam sendo insuficientes e por muitas vezes causam o término prematuro da simulação. Dessa forma, com $S_{min} > 1$, mesmo durante a etapa final de execução, S_{mult} continua ampliando artificialmente o valor da variabilidade V. Idealmente, após tempo suficiente ter passado para que as métricas gerem estimativas mais precisas, S_{mult} precisaria ser desconsiderado no cálculo final. E para isso, $S_{mult} = 1$, o que só pode ser obtido se $S_{min} = 1$.

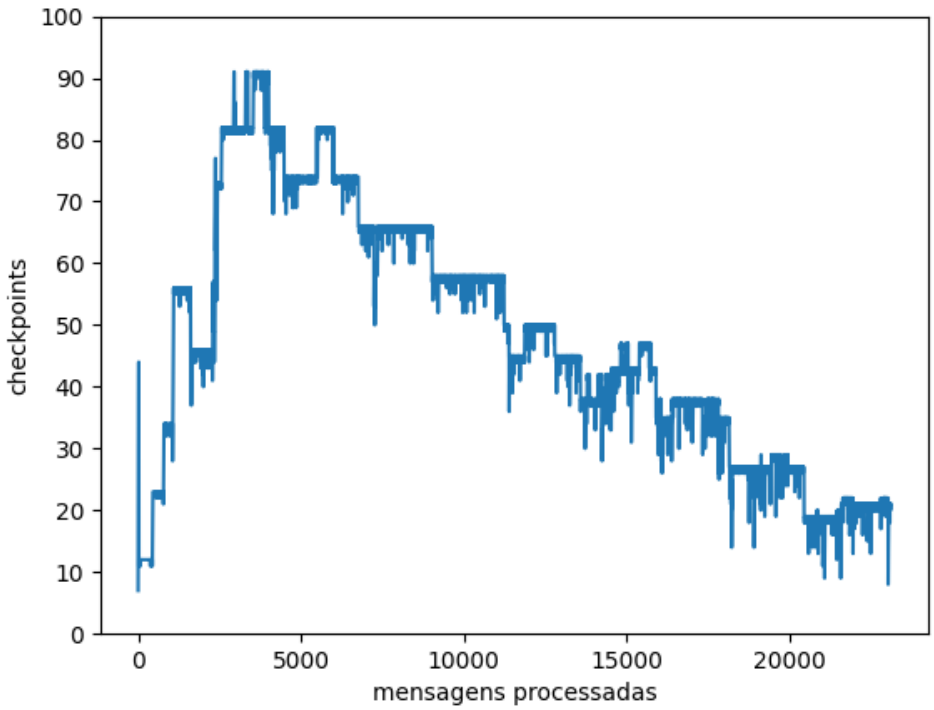

Figura 4 – *Checkpoints* armazenados em um cenário com $S_{min} = 1$

A figura 4 ilustra a quantidade de *checkpoints* armazenados em um cenário que $S_{min} = 1$. A figura 5 exibe o progresso do valor de S_{mult} nesse mesmo cenário, que é inicializado com $S_{init} = 10$. Perceba que quanto mais S_{mult} se aproxima do valor S_{min}, menos *checkpoints* são mantidos. Logo após S_{mult} atingir o valor de S_{min}, a simulação é encerrada prematuramente, pois a quantidade de *checkpoints* armazenada chega a zero.

Dessa forma, conclui-se que as métricas atuais não são suficientes para gerar boas estimativas. Para que isso aconteça, S_{mult} não pode interferir na etapa final de execução. Ou seja, as métricas sozinhas devem ser capazes de estimar uma quantidade suficiente de *checkpoints* a fim de evitar o término prematuro da simulação.

Este trabalho apresenta a integração de uma nova métrica e um novo parâmetro para tornar a estimativa de *checkpoints* mais precisa e não-dependente do multiplicador de segurança

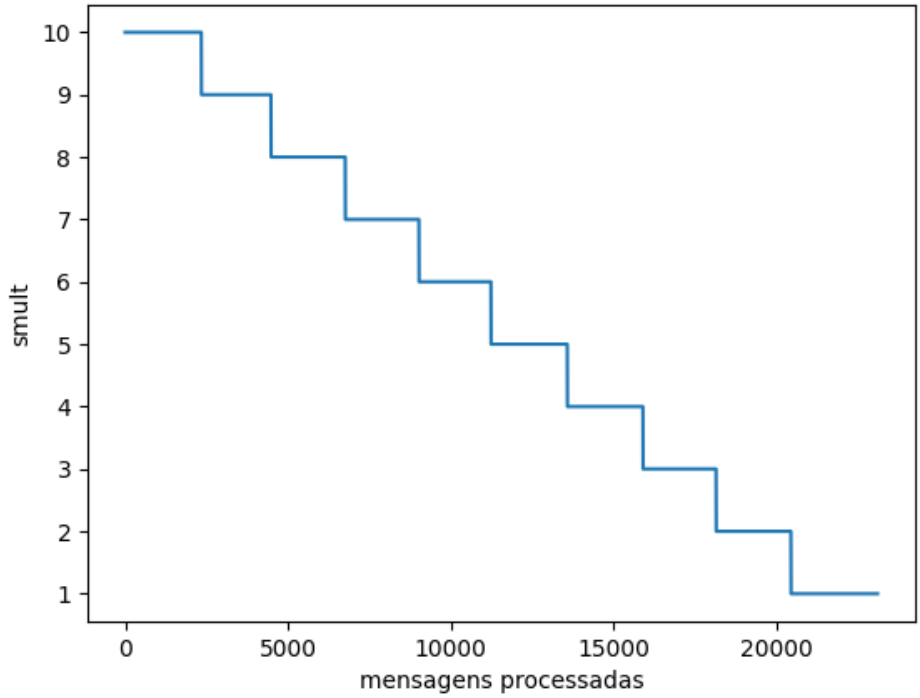

Figura 5 – Progressão de S_{mult} com $S_{min} = 1$

S_{mult} na etapa final de execução. Os detalhes da solução proposta neste trabalho são apresentados na seção seguinte.

3.2 SOLUÇÃO PROPOSTA

A solução proposta neste trabalho partiu do objetivo de complementar as métricas do trabalho original para melhorar a precisão das estimativas. Assim, foi decidido que não haveria uma reformulação drástica no cálculo das estimativas, buscando preservar a ideia original o máximo possível. Neste trabalho, uma nova métrica e um novo parâmetro foram propostos.

O novo parâmetro originou-se da observação de que o RPGC não monitora a porcentagem de crescimento médio de R_{max}. Isto é, apesar de R_{max} ser utilizado como estimativa base, não está sendo levado em consideração o quanto esse valor tem aumentado em relação aos valores anteriores. A importância de levar em consideração essa porcentagem é a de tentar prever qual poderia ser o tamanho do próximo maior *rollback* em relação aos anteriores. Assim, seria possível ampliar a estimativa de R_{max}, protegendo a simulação de *rollbacks* cujo tamanho vão além do maior *rollback* registrado até o presente momento.

Seja R_{len} o tamanho de um *rollback* que aconteceu recentemente. R_{max} é apenas atualizado quando $R_{len} > R_{max}$. Quando essa condição é satisfeita, a porcentagem de crescimento é calculada por $P_{cresc} = R_{len}/R_{max}$. Ou seja, o valor de P_{cresc} informa o quão grande o *rollback* de tamanho R_{len} foi em relação ao maior *rollback* anterior R_{max}. Além disso, o valor de P_{cresc}

é então somado a um acumulador de porcentagens de crescimento P_{accum}, e um contador P_{count} de ocorrências de $R_{len} > R_{max}$ é incrementado.

Assim, a porcentagem de crescimento médio de R_{max} é calculada por:

$$P_{avg} = \frac{P_{accum}}{P_{count}}$$

A nova métrica adicionada aborda a frequência com que os *rollbacks* estão acontecendo. Em cenários que os *rollbacks* estejam acontecendo com mais frequência, se justifica aumentar a quantidade *checkpoints* armazenados. Caso contrário, *rollbacks* de tamanho considerável em sequência podem causar o término prematuro da simulação.

A frequência de *rollbacks* foi adaptada de (QUAGLIA, 2001)(10). Considere R_{count} a quantidade de *rollbacks* que ocorreram desde o início da simulação, e M_{count} a quantidade de mensagens processadas. A frequência de *rollbacks* é dada por:

$$F_{roll} = \frac{R_{count}}{M_{count}}$$

A estrutura da fórmula para cálculo da quantidade de *checkpoints* permanece semelhante a original. As duas principais diferenças são: R_{max} é substituído por $R_{next} = R_{max} \times P_{avg}$ para realizar os mesmos cálculos, porém agora baseados em um possível *rollback* de maior tamanho futuro; e F_{roll} é aplicado na estimativa final de *checkpoints*, ampliando-a proporcional a frequência de *rollbacks* observada.

Assim, o cálculo da variabilidade V é agora dado por:

$$V = max(V_{min}, R_{next} - \lfloor R_{avg} \rfloor)$$

E o cálculo da quantidade de *checkpoints* C_{keep} foi alterado para:

$$C_{keep} = max(C_{min}, R_{next} + S) \times (1.0 + F_{roll})$$

Através destas alterações no cálculo das estimativas, as previsões se tornam não-dependentes do multiplicador de segurança S_{mult} na etapa final de execução, permitindo agora S_{mult} alcançar um valor de $S_{min} = 1$.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentadas, em pseudocódigo, as alteração implementadas no DCB para integração da nova métrica e do novo parâmetro ao RPGC.

O algoritmo 1 é executado sempre após a ocorrência de um *rollback*. Primeiramente é verificado se o tamanho do *rollback* atual foi maior que o tamanho do maior *rollback* previamente registrado. Apenas se essa condição for satisfeita o bloco de código que atualiza o valor de P_{avg} é executado. A linha 2 calcula a porcentagem de crescimento de R_{len} em relação a R_{max} e armazena em P_{cresc}. Em seguida, o contador P_{count} de ocorrências de $R_{len} > R_{max}$ é

incrementado. Na linha 4, o acumulador de tamanhos de *rollback* P_{accum} é somado com R_{len}. Finalmente, o valor de P_{avg} é atualizado com base nos novos valores de P_{accum} e P_{count}, e o tamanho do maior *rollback* armazenado em R_{max} é substituído pelo valor de R_{len}.

O algoritmo 2 realiza o novo cálculo da quantidade de *checkpoints* a serem mantidos. Na linha 2, P_{avg} é utilizado para ampliar o valor de R_{max}, cujo resultado é armazenado em R_{next}. Em seguida, a variabilidade V é calculada pela diferença entre R_{next} e a média de tamanhos de *rollback* R_{avg}. Na linha 4, o valor do multiplicador de segurança S_{mult} é calculado, que multiplica a variabilidade V na linha 5 para determinar o valor da margem de segurança S. Na linha 6, a frequência de *rollbacks* é calculada. Na linha 7, a estimativa da quantidade de *checkpoints* é calculada pela soma de R_{next} com a margem de segurança S. O resultado final é ampliado por $(1.0 + F_{roll})$, que é retornado pela função como a quantidade de *checkpoints* a serem mantidos na simulação.

Algoritmo 1 – Calculo de P_{avg} executado apos um *rollback*

```
1 if R_len > R_max then
2 |   P_cresc ← max(1.0, R_len / R_max);
3 |   P_count ← P_count + 1;
4 |   P_accum ← P_accum + P_cresc;
5 |   P_avg ← P_accum / P_count;
6 |   R_max = R_len;
7 end
```

Algoritmo 2 – Cálculo do RPGC de quantos *checkpoints* devem ser mantidos com base nas métricas coletadas

```
1 Function QuantidadeDeCheckpointsParaManter():
2 |   R_next ← R_max × max(1.0, P_avg);
3 |   V ← max(V_min, R_next − ⌊R_avg⌋);
4 |   S_mult ← max(S_init − (M_rec − T_wait) / S_dec, S_min);
5 |   S ← ⌊V × S_mult⌋;
6 |   F_roll ← R_count / M_count;
7 |   C_keep ← max(C_min, R_next + S) × (1.0 + F_roll);
8 return C_keep
```

4 EXPERIMENTAÇÃO, RESULTADOS E ANÁLISES

Este capítulo apresenta como os casos de teste foram elaborados para o algoritmo antigo e para a nova implementação. Em seguida, os resultados dos testes são apresentados e uma análise comparativa é realizada.

4.1 ESTUDOS DE CASO

Para os estudos de caso foi montado um modelo igual ao utilizado em (ADAMCZUK, 2021)(1). O modelo é composto por três processos lógicos $P1$, $P2$ e $P3$, de modo que $P1$ envia mensagens para $P2$, $P2$ envie mensagens para $P3$, que por sua vez envia mensagens para $P1$. Isto torna o modelo simétrico permitindo que seja possível analisar o modelo como um todo sendo necessário apenas observar o comportamento de um processo. A figura 6 ilustra como o modelo foi montado.

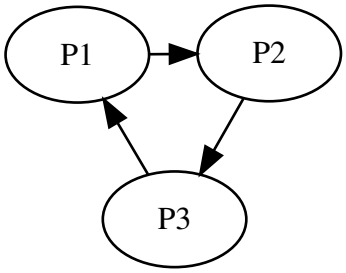

Figura 6 – Modelo utilizado nos estudos de caso

Para cada teste realizado foi aplicado um padrão de envio de mensagens diferente, que ocorre através da manipulação do *timestamp* de execução das mensagens enviadas. Essa estratégia permite a manipulação dos padrões de *rollback* da simulação, alterando o tamanho dos *rollbacks* e a frequência com que ocorrem. O *timestamp* de execução é calculado através de uma função $rand(\mu, \sigma)$ que retorna um número aleatório segundo a distribuição normal $N(\mu, \sigma^2)$. O valor gerado é somado ao LVT atual do componente remetente, que resulta no *timestamp* de execução da mensagem enviada para o componente de destino. Mais especificamente, $LVT + max(3, rand(\mu, \sigma))$. Assim, quanto maior o valor de σ, maior o tamanho máximo dos *rollbacks*, e consequentemente, maior a variabilidade.

É importante ressaltar que o valor de σ também afeta a distância das mensagens enviadas, o que não apenas afeta a quantidade de mensagens totais processadas, como também afeta o avanço do tempo de simulação. A figura 7 exibe uma simulação executada com uma distribuição $N(100, 10^2)$ e outra simulação executada com uma distribuição $N(100, 100^2)$. Em ambos os exemplos, a execução encerrava quando ao menos um dos processos chegasse a marca de 10^5 mensagens recebidas. Na simulação com $\sigma = 10$ o avanço do tempo de simulação e a quantidade de mensagens processadas foi muito maior que o da simulação com $\sigma = 100$.

Assim, uma distância menor entre as mensagens resulta em um maior número de mensagens processadas e em um avanço maior de tempo de simulação.

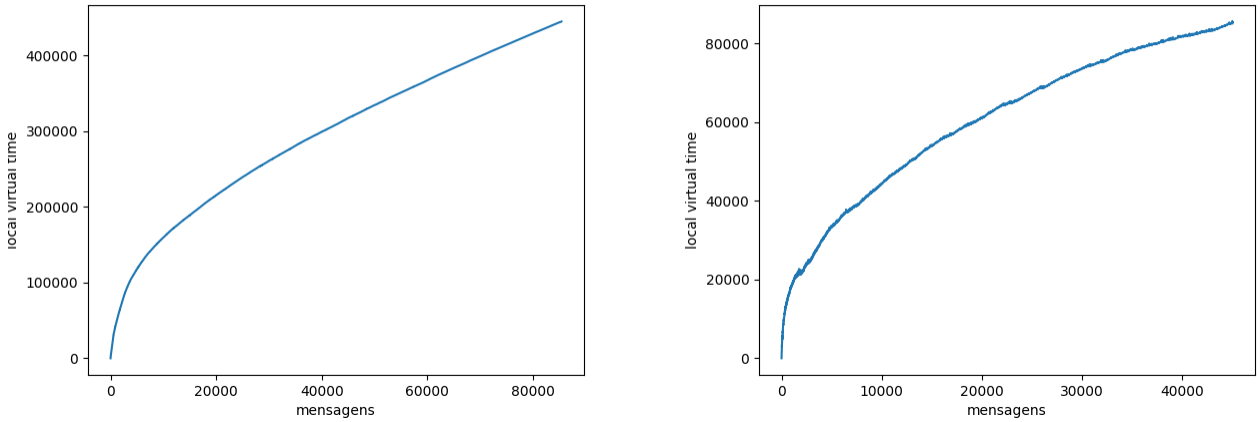

Figura 7 – Avanço de *lvt* (esquerda: $\sigma = 10$, direita: $\sigma = 100$)

Foram realizados cinco testes para a implementação de (ADAMCZUK, 2021)(1) e cinco testes para a implementação apresentada neste trabalho. Cada teste apresenta um valor de σ diferente para a distribuição $N(\mu, \sigma^2)$, como apresentado na tabela 4. Em todos os testes, cada um dos três processos $P1$, $P2$ e $P3$ recebeu 10^5 mensagens (entretanto não implica que as 10^5 mensagens foram processadas).

Para cada mensagem processada, um *checkpoint* é criado. Isto torna os *rollbacks* mais perceptíveis nos gráficos ilustrando a quantidade de *checkpoints*.

Nas configurações das constantes do RPGC, também foram utilizados os mesmos valores dos testes apresentados em (ADAMCZUK, 2021)(1): $C_{min} = 10$; $S_{init} = 10$; $S_{dec} = 3000$; $T_{wait} = 30$ e $W = 0.05$. A única exceção é o valor mínimo do multiplicador de segurança, que no trabalho original é configurado como $S_{min} = 4$, e neste trabalho é configurado como $S_{min} = 1$.

Teste	Distribuição
T_1	$N(100, 200^2)$
T_2	$N(100, 100^2)$
T_3	$N(100, 50^2)$
T_4	$N(100, 20^2)$
T_5	$N(100, 10^2)$

Tabela 4 – Distribuição utilizada em cada teste

4.2 RESULTADOS E ANÁLISES

Iniciamos a análise dos resultados comparando a quantidade média de *checkpoints* armazenados em cada teste. Após isso é feito uma análise sobre a nova métrica e o novo parâmetro apresentados neste trabalho, e em seguida algumas observações sobre o trabalho original. Todos os resultados exibidos abaixo são referentes ao processo lógico P_1 dos estudos de caso.

4.2.1 Quantidade média de *checkpoints*

A quantidade média de *checkpoints* foi calculada levando em consideração a quantidade de *checkpoints* armazenados desde o início da simulação. O motivo é que o trabalho original depende de um valor de $S_{mult} > 1$ mesmo após o término da segunda etapa de execução do RPGC. As métricas propostas por este trabalho permitem que S_{mult} assuma o valor de $S_{min} = 1$, de modo que o cálculo das estimativas dependam apenas do valor das métricas.

Os resultados são exibidos na tabela 5. Perceba que a redução na quantidade de *checkpoints* tende a variar entre os testes, porém se mantendo sempre positiva.

Teste	Distribuição	Este trabalho	Trabalho original	Redução %
T_1	$N(100, 200^2)$	237	330	28
T_2	$N(100, 100^2)$	204	257	20
T_3	$N(100, 50^2)$	136	144	5
T_4	$N(100, 20^2)$	57	79	27
T_5	$N(100, 10^2)$	40	43	6

Tabela 5 – Comparativo da quantidade de *checkpoints* para cada teste realizado

A figura 8 exibe a progressão da quantidade média de *checkpoints* para o teste T_5. A figura à esquerda representa os resultados obtidos com o trabalho original, e a figura à direita os resultados obtidos pelas mudanças realizadas neste trabalho. Perceba que a quantidade de *checkpoints* armazenada foi maior no início da simulação para este trabalho. Isto ocorre devido à compensação realizada pela nova métrica e parâmetro adicionados ao cálculo das estimativas. Entretanto, a partir do momento que o multiplicador de segurança S_{mult} vai atingindo valores menores, a quantidade de *checkpoints* armazenados diminui proporcionalmente.

Como neste trabalho o multiplicador de segurança consegue atingir um valor de $S_{min} = 1$ na terceira etapa de execução, a quantidade de *checkpoints* armazenada tende a ser sempre menor a partir desse instante, pois a estimativa depende agora apenas dos valores das métricas. Isso fica evidente observando a figura 9. Para o trabalho antigo, o menor valor de S_{mult} atingido é o de $S_{min} = 4$, próximo da marca de 20.000 mensagens processadas. A partir desse instante o valor de S_{mult} fica estabilizado nesse valor. Como é possível observar pela figura 8, é exatamente

a partir desse momento que a quantidade de *checkpoints* parou de reduzir no trabalho original. Para este trabalho, S_{mult} estabilizou no valor de $S_{min} = 1$ após 20.000 mensagens processadas, assim tendo uma duração da segunda etapa um pouco mais prolongada em comparação. Porém, a partir desse instante as estimativas são realizadas sem a interferência do multiplicador de segurança S_{mult}, tendendo a manter sempre uma quantidade menor de *checkpoints* em relação ao trabalho original.

Já na figura 10 é ilustrado o impacto das alterações para o cálculo da estimativa final de *checkpoints* no teste T_5. Cada linha desta figura representa o valor anterior da estimativa combinado com o valor de uma outra métrica ou parâmetro adicional. Começando com o valor da estimativa base, o tamanho do maior *rollback* R_{max}, multiplicando-o com a porcentagem de crescimento P_{avg}, somando esse valor com a variabilidade (incluindo a multiplicação com S_{mult}), e por fim ampliando o valor final com a frequência de *rollbacks*. A linha laranja mais acima então, representa o valor final da estimativa. Para o teste T_5 percebe-se que a frequência de *rollbacks* não teve um impacto muito significativo no resultado final, apenas ampliando a quantidade de *checkpoints* consideravelmente. Já a porcentagem de crescimento P_{avg} teve um papel importante em aumentar tanto a estimativa base R_{max} quanto o valor da variabilidade V, que constituem a maior parte do resultado final.

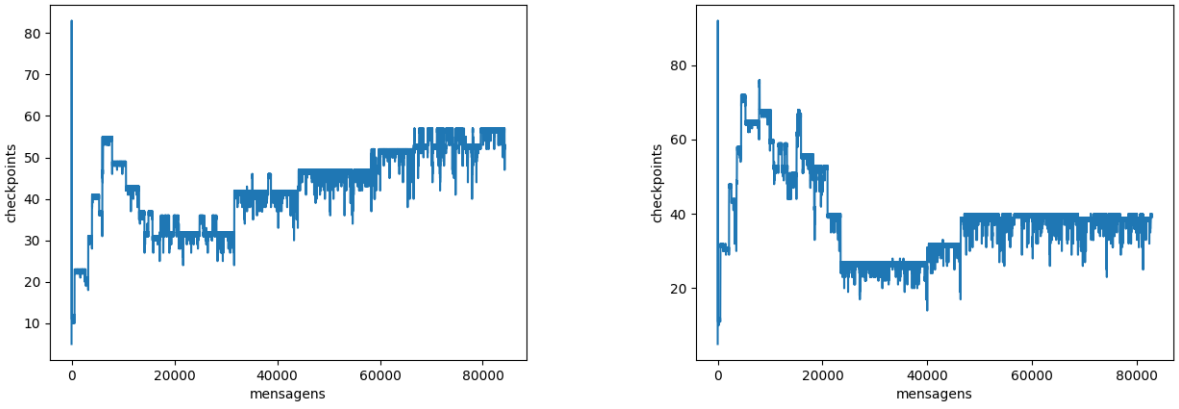

Figura 8 – Evolução de *checkpoints* para teste T_5 (esquerda: trabalho original, direita: este trabalho)

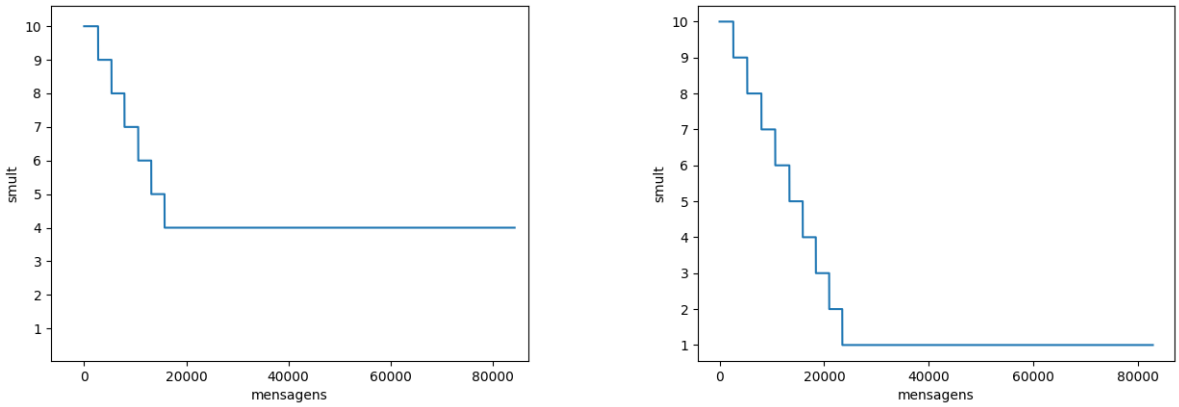

Figura 9 – Evolução de S_{mult} para teste T_5 (esquerda: trabalho original, direita: este trabalho)

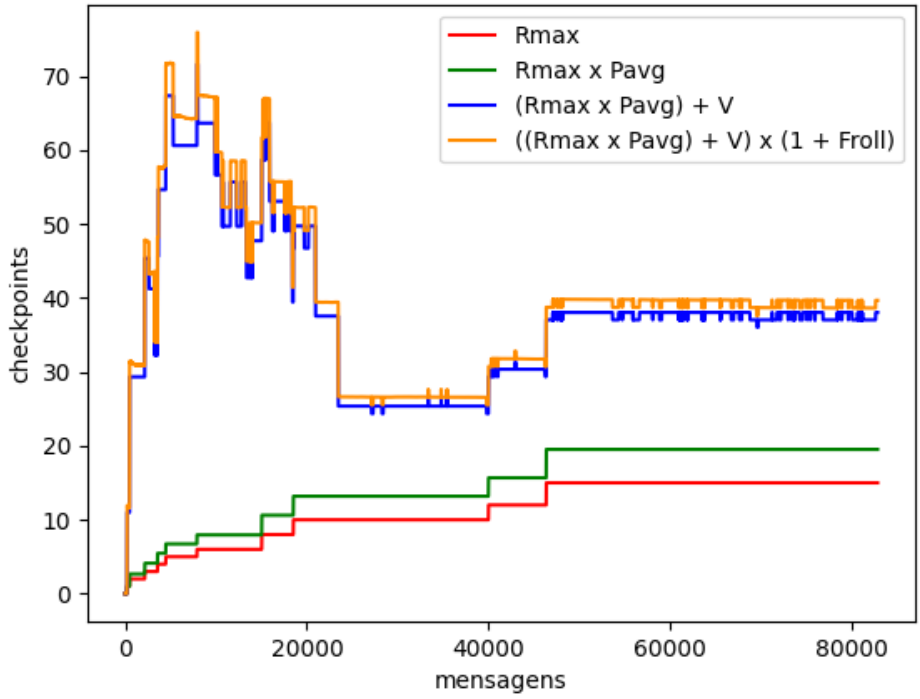

Figura 10 – Influência das métricas e parâmetros combinados para o teste T_5

Para o teste T_1, a progressão da quantidade média de *checkpoints* é mostrada na figura 11. Como ocorreu no teste T_5, a tendência deste trabalho é a de manter mais *checkpoints* nas primeiras etapas de execução devido a presença de uma nova métrica e de um novo parâmetro. Percebe-se que quanto maior o tamanho dos *rollbacks* durante o início da execução, mais desproporcional essa quantidade tende a ser em relação ao trabalho original. Porém, como observado anteriormente, esse valor diminui conforme S_{mult} vai atingindo valores menores.

Já a progressão de S_{mult} ocorreu de forma diferente em relação ao teste T_5, como mostra a figura 12. No teste T_1, o multiplicador de segurança S_{mult} atingiu o valor de S_{min} por volta de 10.000 mensagens processadas, enquanto no teste T_5, o valor de S_{mult} estabilizou por volta de 20.000 mensagens processadas. Isto ocorre devido a distância média das mensagens enviadas ser menor em T_5 com $\sigma = 10$ em relação a T_1 com $\sigma = 200$. Assim, mais mensagens são enviadas e processadas no teste T_5 devido a distância média entre as mensagens ser menor. Como o decréscimo de S_{mult} ocorre por múltiplos de S_{dec} mensagens recebidas, naturalmente S_{mult} atingirá S_{min} em instantes diferentes dependendo da distância média entre as mensagens.

Além disso, uma distância média maior entre as mensagens afeta diretamente o cálculo da frequência de *rollbacks*, que passa a assumir valores maiores. Na figura 13 pode-se observar como a frequência de *rollbacks* teve um impacto maior no teste T_1 em comparação ao teste T_5, ampliando o valor final de maneira considerável tanto no início como no final da execução.

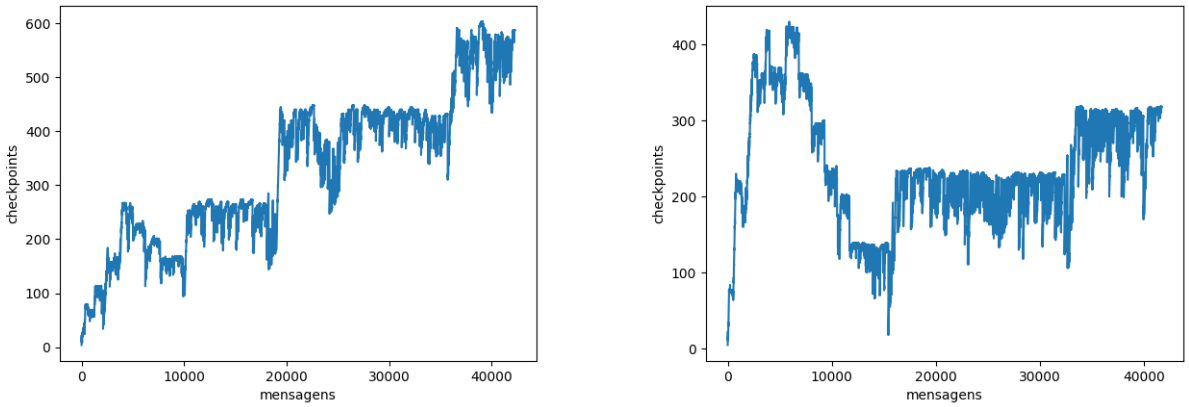

Figura 11 – Evolução de *checkpoints* para teste T_1 (esquerda: trabalho original, direita: este trabalho)

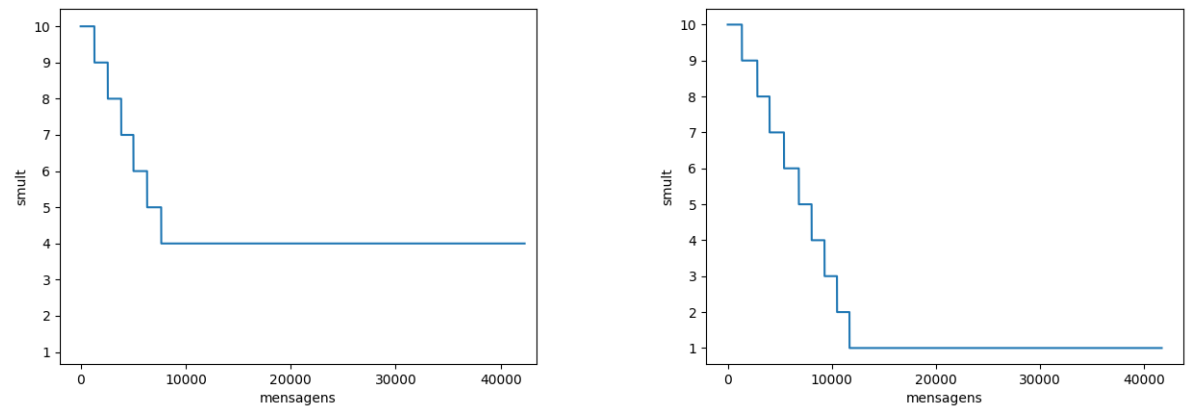

Figura 12 – Evolução de S_{mult} para teste T_1 (esquerda: trabalho original, direita: este trabalho)

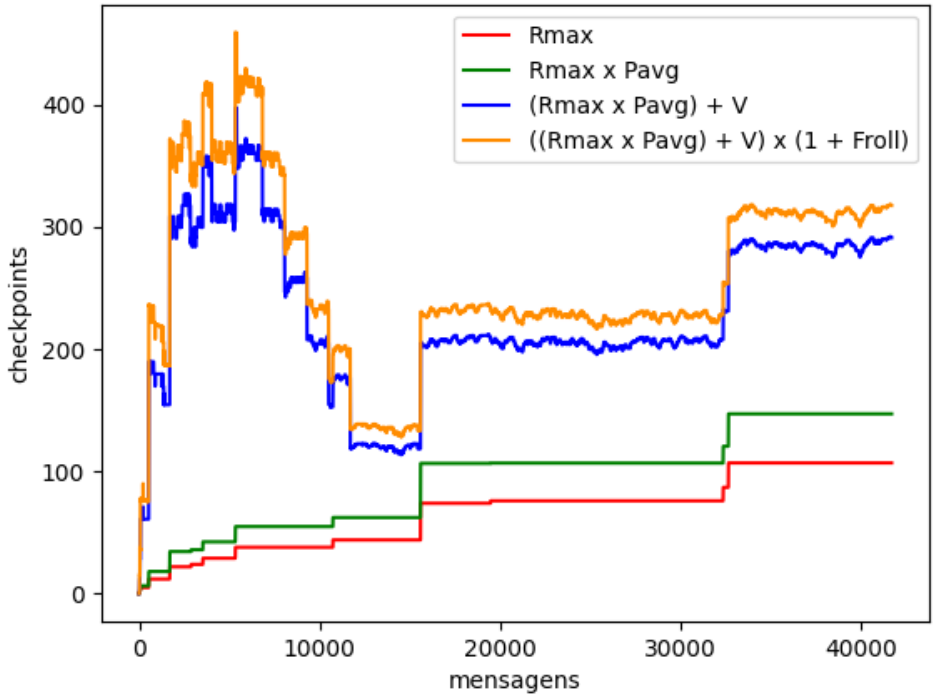

Figura 13 – Influência das métricas e parâmetros combinados para o teste T_1

4.2.2 Frequência de *rollbacks*

Como observado anteriormente, quanto menor a distância entre as mensagens, maior a quantidade de mensagens processadas. Consequentemente, a frequência de *rollbacks* acabou assumindo valores pequenos nos testes T_5, T_4 e T_3, como pode ser observado na tabela 6. Em cenários em que a distância entre as mensagens foi maior, a frequência de *rollbacks* acabou assumindo valores maiores. Nos testes T_2 e T_1 houve uma ampliação média de 106% e 123% na estimativa final, respectivamente.

Além disso, em alguns testes executados sem a inclusão da frequência de *rollbacks*, a simulação encerrava prematuramente quando a distância entre as mensagens era alta. Assim, conclui-se que a frequência de *rollbacks* teve um papel importante nesses cenários.

No entanto, pode-se observar pela figura 14, referente a progressão de F_{roll} no teste T_1, que o valor da frequência de *rollbacks* tende a estabilizar em um valor relativamente pequeno. Isso acontece devido ao cálculo atual utilizar o intervalo de todo o tempo de simulação. Seria possível obter valores mais relevantes se fosse apenas considerado um intervalo menor de um passado recente.

Teste	Distribuição	F_{roll}
T_1	$N(100, 200^2)$	0,123
T_2	$N(100, 100^2)$	0,106
T_3	$N(100, 50^2)$	0,086
T_4	$N(100, 20^2)$	0,060
T_5	$N(100, 10^2)$	0,048

Tabela 6 – Comparativo do valor médio de F_{roll} para cada teste realizado

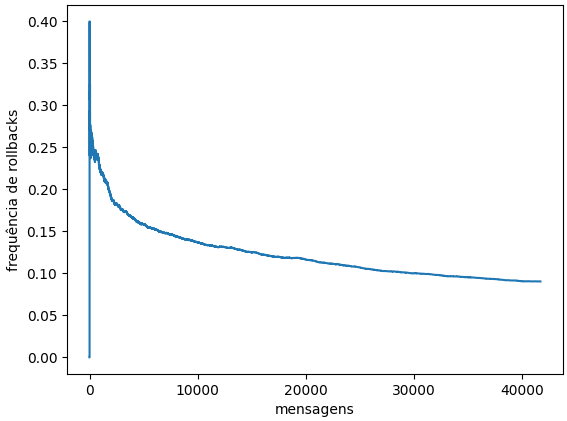

Figura 14 – Progresso de F_{roll} para o teste T_1

4.2.3 Porcentagem média de crescimento do maior *rollback*

A tabela 7 exibe os valores que porcentagem média de crescimento do maior *rollback* assumiu nos testes. Em média, os maiores *rollbacks* têm crescido por um fator entre 1,31 e 1,46 do maior tamanho anterior.

Teste	Distribuição	P_{avg}
T_1	$N(100, 200^2)$	1,41
T_2	$N(100, 100^2)$	1,45
T_3	$N(100, 50^2)$	1,46
T_4	$N(100, 20^2)$	1,34
T_5	$N(100, 10^2)$	1,31

Tabela 7 – Comparativo do valor de P_{avg} para cada teste realizado

A figura 15, referente ao teste T_1, mostra qual foi a precisão da estimativa de tamanho do próximo maior *rollback* a acontecer. A linha verde exibe qual a estimativa calculada por $R_{max} \times P_{avg}$ desde a ocorrência do último maior *rollback*. A linha vermelha o maior *rollback* que acabou de acontecer.

Em outras palavras, em instantes que a linha verde está acima da linha vermelha, significa que P_{avg} conseguiu realizar uma boa estimativa de o quão grande o próximo *rollback* seria. Como se pode observar, no início as estimativas tendem a ser menos precisas, mas conforme *rollbacks* de tamanhos maiores acontecem, as estimativas tendem a melhorar.

Já a figura 16 exibe a progressão do valor de P_{avg} para o mesmo teste T_1. Em instantes que a linha sobe no gráfico, foram registrados *rollbacks* com proporção acima da média de crescimento. Os valores iniciais tendem a ser maiores e menos precisos, porém estabilizam em um valor relativamente menor do que no início.

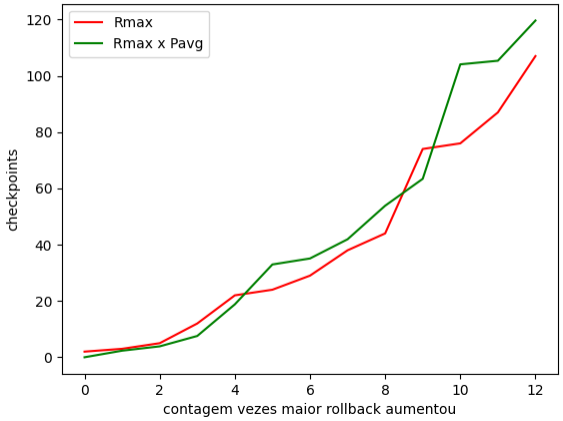

Figura 15 – Diferença entre $R_{max} \times P_{avg}$ e R_{max} para o teste T_1

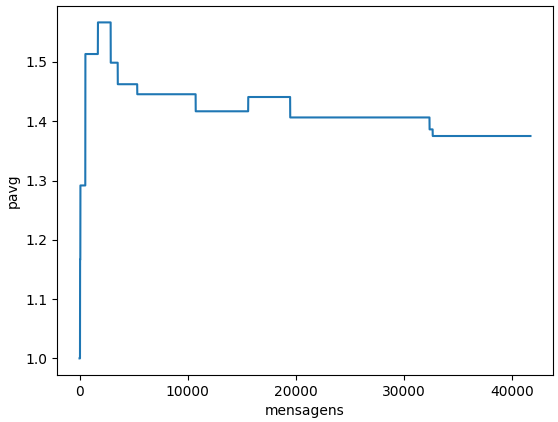

Figura 16 – Progresso de P_{avg} para o teste T_1

4.2.4 Variabilidade

Uma fragilidade foi observada em relação à variabilidade V. Há situações em que a previsão da quantidade *checkpoints* deste trabalho se tornam insuficientes, causando o término da simulação. Em cenários que a variabilidade assume valores pequenos após $S_{mult} = S_{min}$, as estimativas não são capazes de resistir a *rollbacks* grandes em sequência.

A variabilidade assume valores menores quando a média de *rollbacks* R_{avg} e o tamanho do maior *rollback* R_{max} possuem valores muito próximos. Isto gera um cenário arriscado, pois menos *checkpoints* são mantidos além do maior tamanho de *rollback*. Esse cenário tende a ocorrer com mais frequência quando a distância entre as mensagens é menor. A figura 17 ilustra esse cenário. Perceba como no final do gráfico a quantidade de *checkpoints* diminui repentinamente. Isso indica que um *rollback* bem acima do esperado aconteceu. Já na figura 18, é possível observar o que ocorre com o valor da variabilidade nessas situações. O salto repentino registrado no gráfico representa um valor além do previsto pela variabilidade.

Assim, se faz necessário alterações para que em cenários que a variabilidade assuma valores pequenos, a estimativa não seja otimista demais.

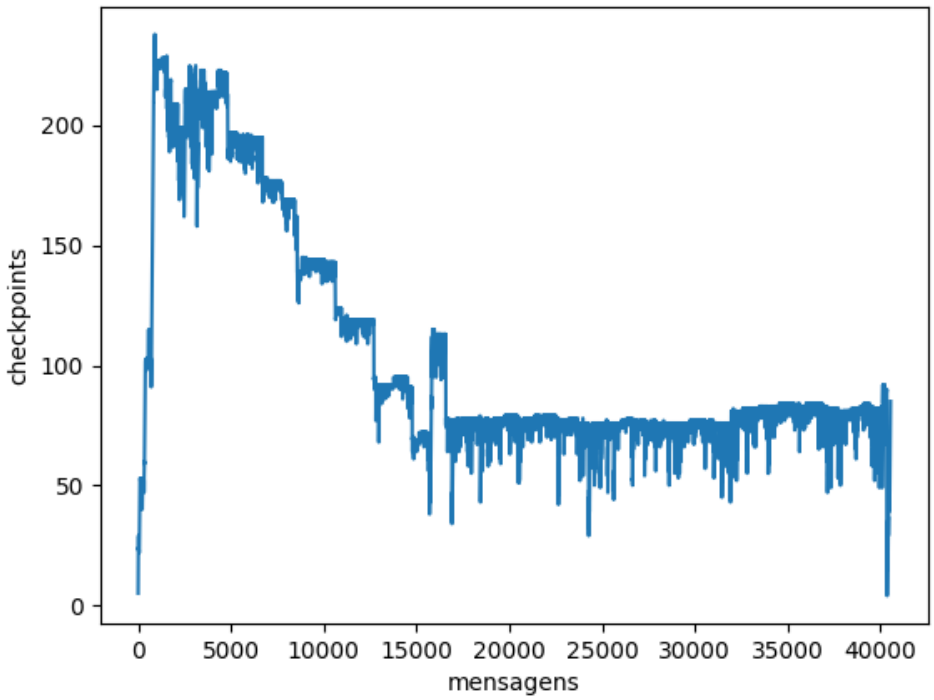

Figura 17 – Progressão na quantidade de *checkpoints* após a ocorrência de um *rollback* desproporcional

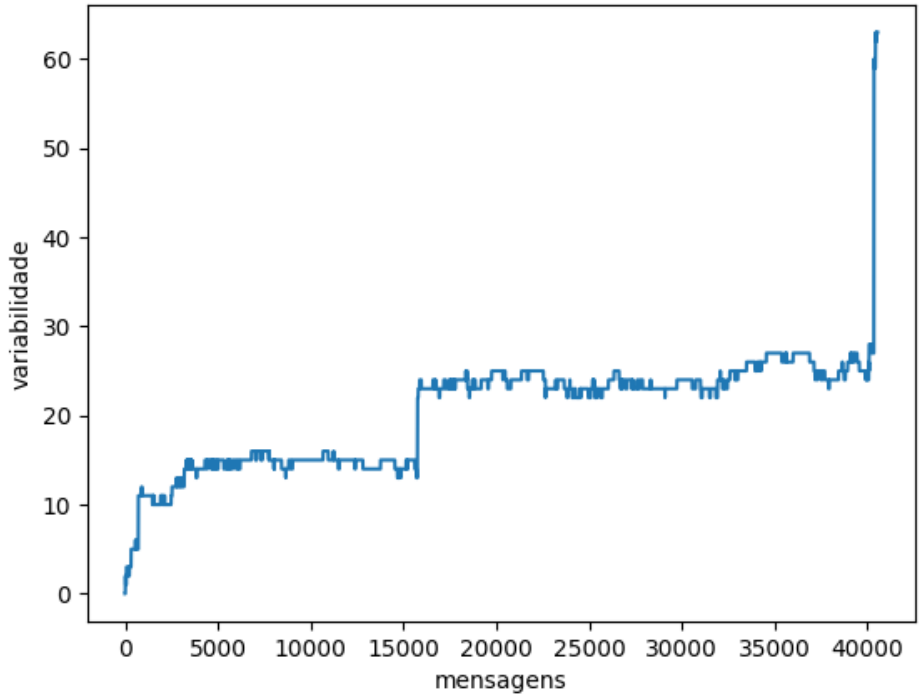

Figura 18 – Progressão da variabilidade em uma simulação que encerrou prematuramente

5 CONCLUSÃO

Este trabalho apresentou a implementação de uma nova métrica e de um novo parâmetro para o *garbage collector* assíncrono RPGC. A frequência de *rollbacks* desempenhou um papel considerável em ampliar a estimativa final em situações que *rollbacks* ocorreram com mais frequência. A porcentagem de crescimento médio do maior *rollback* apresentou resultados satisfatórios em prever o quanto os maiores *rollbacks* têm aumentado em relação as ocorrências passadas. As alterações foram implementadas e testadas na arquitetura DCB. Os resultados demonstraram que o cálculo das estimativas se tornou mais preciso. Além disso, as estimativas se tornaram não-dependentes do multiplicador de segurança S_{mult} na etapa final de execução.

5.1 TRABALHOS FUTUROS

Mudanças podem ser realizadas no cálculo da variabilidade, principalmente quando assume valores pequenos. Uma nova métrica ou um novo parâmetro complementar poderiam ser implementados, assim como uma reformulação no cálculo da variabilidade poderia ser realizada.

Atualmente o RPGC não identifica a dependência de *checkpoints* entre os componentes da simulação, tornando-o suscetível ao efeito cascata. A propagação de um vetor de dependências junto às mensagens de aplicação pode resolver esse problema.

O cálculo da frequência de *rollbacks* pode ser alterado de modo a utilizar apenas um intervalo recente da simulação. Atualmente o intervalo de todo o tempo de simulação é utilizado. Concentrar a coleta de informações à um intervalo de tamanho particular pode resultar em estimativas mais precisas.

REFERÊNCIAS

1 ADAMCZUK, Marcos Theophilo Gobbi. Garbage Collector Assíncrono em Simulação Distribuída, 2021.

2 ELNOZAHY, Elmootazbellah Nabil et al. A survey of rollback-recovery protocols in message-passing systems. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 34, n. 3, p. 375–408, 2002.

3 FUJIMOTO, Richard M. **Parallel and distributed simulation systems**. [S.l.]: Citeseer, 2000. v. 300.

4 JEFFERSON, David R. Virtual time. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, ACM New York, NY, USA, v. 7, n. 3, p. 404–425, 1985.

5 MATTERN, Friedemann. Efficient algorithms for distributed snapshots and global virtual time approximation. **Journal of parallel and distributed computing**, Elsevier, v. 18, n. 4, p. 423–434, 1993.

6 MELLO, Braulio Adriano de. Co-simulação distribuıda de sistemas heterogêneos, 2005.

7 PARIZOTTO, Ricardo; MELLO, Braulio de. Uma abordagem para minimizar pontos de verificação inúteis em simulações otimistas distribuídas. In: ANAIS do XLVI Seminário Integrado de Software e Hardware. Belém: SBC, 2019. p. 12–21. DOI: 10.5753/semish.2019.6563. Disponível em: <https://sol.sbc.org.br/index.php/semish/article/view/6563>.

8 PIDD, Michael. An introduction to computer simulation. In: IEEE. PROCEEDINGS of Winter Simulation Conference. [S.l.: s.n.], 1994. p. 7–14.

9 PUTTLITZ, Cleiton. Intervalo adaptativo entre checkpoints em simulação distribuída, 2021. Disponível em: <https://rd.uffs.edu.br/handle/prefix/4650>.

10 QUAGLIA, Francesco. A cost model for selecting checkpoint positions in Time Warp parallel simulation. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 12, n. 4, p. 346–362, 2001.

11 SARASWAT, Bal Krishna; SURYAVANSHI, R; YADAV, DS. A comparative study of checkpointing algorithms for distributed systems. **Int. J. Pure Appl. Math**, v. 118, p. 1595–1603, 2018.

12 SCHMIDT, Rodrigo et al. Optimal asynchronous garbage collection for RDT checkpointing protocols. In: IEEE. 25TH IEEE International Conference on Distributed Computing Systems (ICDCS'05). [S.l.: s.n.], 2005. p. 167–176.

13 WANG, Yi-Min. Consistent global checkpoints that contain a given set of local checkpoints. **IEEE Transactions on Computers**, IEEE, v. 46, n. 4, p. 456–468, 1997.