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

PAULO GABRIEL SENA COMASETTO

ESCALONAMENTO DE MENSAGENS PENDENTES COM SINCRONIZAÇÃO HÍBRIDA ENTRE PROCESSOS LÓGICOS

CHAPECÓ
2023

PAULO GABRIEL SENA COMASETTO

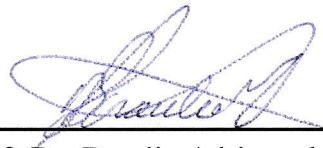
ESCALONAMENTO DE MENSAGENS PENDENTES COM SINCRONIZAÇÃO HÍBRIDA ENTRE PROCESSOS LÓGICOS

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

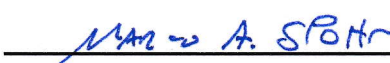
Orientador: Prof. Dr. Braulio Adriano de Mello
Coorientador: Me. Ricardo Parizotto

Este trabalho de conclusão de curso foi defendido e aprovado pela banca avaliadora em: 13/02/2023.

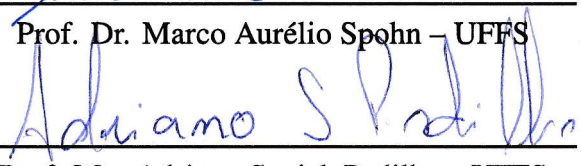
BANCA AVALIADORA



Prof. Dr. Braulio Adriano de Mello – UFFS



Prof. Dr. Marco Aurélio Spohn – UFFS



Prof. Me. Adriano Sanick Padilha – UFFS

Dedico este trabalho ao meu colega, Vinicius Cornelius, amigo incondicional e de coração, e que hoje zela por todos nós. *Per aspera, ad astra.*

“The Grid. A digital frontier. I tried to picture clusters of information as they moved through the computer. What did they look like? Ships, motorcycles? Were the circuits like freeways? I kept dreaming of a world I thought I’d never see. And then, one day... I got in.”

(Kevin Flynn)

Escalonamento de mensagens pendentes com sincronização híbrida entre Processos Lógicos

Paulo Gabriel Sena Comasetto[1], Braulio Adriano de Mello[1]*, Ricardo Parizotto[2]†

[1] Curso de Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS)
Chapecó – Santa Catarina – Brasil

[2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – Rio Grande do Sul – Brasil

paulogscomasetto@gmail.com, braulio@uffs.edu.br, rparizotto@inf.ufrgs.br

Abstract. *In distributed simulation, synchronization between Logical Processes is performed by conservative or optimistic protocols. The combination of both algorithms in the same simulation, however, leads to the appearance of violations and conflicts, causing loss of performance and precision. In this sense, this work addresses a message scheduling algorithm designed to mitigate hybrid synchronization obstacles. Adjustments and modifications to the algorithm were implemented, followed by its integration into the DCB architecture. Complementing the original numerical tests, simulations were run on the platform, and the results showed advantages in using the proposed solution compared to the LTF policy. Future perspectives include tests with real models, use of other performance metrics and comparison with other scheduling algorithms.*

Resumo. *Em simulação distribuída, a sincronização entre os Processos Lógicos é realizada por protocolos conservadores ou otimistas. A combinação de ambos os algoritmos em uma mesma simulação, no entanto, propicia o surgimento de violações e conflitos, ocasionando perda de desempenho e precisão. Nesse sentido, o presente trabalho aborda um algoritmo escalonador de mensagens projetado para mitigar obstáculos da sincronização híbrida. Foram implementados ajustes e modificações no algoritmo, seguidos da sua integração na arquitetura DCB. Complementando os testes numéricos originais, simulações foram rodadas na plataforma, e os resultados apontaram vantagens no uso da solução proposta comparado à política LTF. Perspectivas futuras incluem testes com modelos reais, utilização de outras métricas de desempenho e cotejo com outros algoritmos de escalonamento.*

1. Introdução

Modelagem e Simulação (M&S) é uma área da Ciência da Computação que trata da representação e execução computacional de sistemas reais ou tangíveis [Carson 2004, Pidd 1994]. A intenção é simplificar o sistema ou processo real através da construção de um modelo, o qual incorpora somente os aspectos mais relevantes do sistema tangível [Carson 2004, Pidd 1994, Chwif and Medina 2006]. A simulação, portanto, consiste na

*Orientador.
†Coorientador.

execução temporal desse modelo em um computador digital, com o intuito de suscitar as mudanças e interações entre os componentes do modelo e observar o seu comportamento ao longo do tempo [Carson 2004, Pidd 1994].

O ato de descentralizar uma simulação, repartindo-a entre múltiplas máquinas independentes e heterogêneas, configura uma Simulação Distribuída (DS). Em DS, as partes do modelo são denominadas Processos Lógicos (PL) ou simuladores, e se comunicam através da troca de mensagens (eventos) com carimbo de tempo [Fujimoto 2015]. É provado que, se todos os PLs processarem os eventos em ordem de estampa temporal — fenômeno intitulado *Local Causality Constraint* (LCC) —, a execução distribuída da simulação terá os mesmos resultados caso a execução tivesse sido sequencial [Fujimoto 2000]. Se não for possível acatar a LCC, um evento pode gerar consequências no passado de outro, pondo em risco o valimento da simulação [Fujimoto 2000, Fujimoto 2003].

Em simulação descentralizada, a natureza distribuída e autônoma dos simuladores impossibilita antecipar violações temporais oriundas das interações entre PLs [Fujimoto 2000, Fujimoto 2003, Lindén 2018]. Esse entrave é abordado por meio de algoritmos de sincronização, que visam assegurar a observância da LCC. Técnicas conservadoras (síncronas) buscam evitar a ocorrência de violações de causalidade, por meio de um mecanismo de troca de *lookaheads* entre os PLs. Por outro lado, métodos de execução otimistas (assíncronos) permitem que PLs processem eventos livremente, mas prescrevem dispositivos de *rollback* (reversão) e restauração para lidar com violações causais que surgirem [Fujimoto 2003, Lindén 2018].

Uma abordagem promissora, denominada "sincronização híbrida", consiste em combinar PLs síncronos e assíncronos em uma mesma simulação [Parizotto and Mello 2022, Jefferson and Barnes 2017, Eker et al. 2021]. Entretanto, coadunar PLs com diferentes protocolos de sincronização é desafiador. Levando em conta que cada evento é executado durante um período de tempo simulado, em vez de ser um episódio instantâneo, podem ocorrer conflitos entre mensagens pendentes. Nesses casos, não é possível processar todas as mensagens, e a precisão da simulação é comprometida [Parizotto and Mello 2022].

Em [Parizotto and Mello 2022], foi apresentada uma abordagem que trata a sincronização híbrida como um problema de escalonamento de mensagens trocadas entre PLs. A solução propôs um algoritmo escalonador de mensagens com o intuito de alcançar o equilíbrio entre o máximo de trabalho exercido e o máximo de mensagens processadas. Os autores argumentaram que maximizar o total de mensagens agendadas também reduz o volume de violações temporais na simulação.

A solução supramencionada foi validada com um teste numérico que apontou para uma possível aplicabilidade em simulações reais. A despeito da importância da avaliação numérica, ela não substitui a experimentação em um ambiente real de simulação. Uma plataforma funcional de simulação distribuída permite considerar e visualizar aspectos como: interação entre PLs distribuídos e autônomos, processamento de violações temporais, consumo de recursos com *rollbacks* e *checkpoints*, dentre outros.

Neste trabalho, foram efetuadas modificações e ajustes ao algoritmo de escalonamento original definido em [Parizotto and Mello 2022]. As principais contribuições

foram no emprego de memoização para partes da solução. Posteriormente, o algoritmo foi integrado à implementação da arquitetura de simulação distribuída DCB (*Distributed Co-simulation Backbone*) e testado nesse mesmo ambiente.

Os resultados obtidos pela experimentação na plataforma de simulação mostraram que, especialmente em situações com aumento do tempo total simulado, o algoritmo proposto sobressaiu-se em relação à política LTF (*Lowest Timestamp First*). Foi constatado que a solução proposta foi mais eficiente em maximizar a soma de trabalho realizado, confrontando com o total de mensagens processadas. Testes adicionais para análise do impacto das memoizações propostas neste trabalho manifestaram uma perda de performance ao desligá-las, que poderia se intensificar em simulações de maior porte.

O restante deste documento está organizado da forma como segue. A Seção 2 aborda a explicação de conceitos basilares do trabalho. Na Seção 3, os trabalhos correlatos selecionados são expostos e discutidos. A Seção 4 discorre sobre aspectos do escalonamento de mensagens, detalha os ajustes aplicados no algoritmo original e relata a etapa de integração da solução à plataforma DCB. A definição dos parâmetros, métricas e modelos encontra-se na Seção 5, assim como a análise dos resultados coletados das simulações. Por fim, a Seção 6 abrange as considerações finais e perspectivas futuras do trabalho.

2. Referencial teórico

Neste capítulo serão elencados e explanados alguns conceitos relevantes para o entendimento do trabalho.

2.1. Modelagem e Simulação

Um sistema tangível, também chamado de processo ou sistema real, pode ser definido como um agrupamento de partes operando em conjunto e com um objetivo em comum [Pidd 1994, Chwif and Medina 2006]. A modelagem computacional trata da representação desse sistema em um modelo, no contexto de um computador digital. O propósito do modelo é capturar os aspectos e regras mais relevantes do sistema, de modo a abstrair e criar uma aproximação do processo real. Idealmente, o modelo deve ser mais simples do que o sistema tangível [Pidd 1994, Carson 2004, Chwif and Medina 2006].

De forma complementar, o campo da simulação computacional visa a incorporar o fator temporal a um modelo [Carson 2004]. Em outras palavras, uma simulação é a execução do modelo por um período de tempo, a fim de desencadear mudanças no seu estado e interações dinâmicas entre seus componentes. Dá-se o nome de Modelagem e Simulação (M&S) à junção das duas áreas do conhecimento [Carson 2004, Fujimoto 2000].

A simulação é uma ferramenta útil para testar, avaliar, comparar e analisar alternativas de sistemas de maneira segura e controlada, sem impactos no processo real [Carson 2004]. Também serve para experimentar uma política antes de sua efetivação e, assim, averiguar hipóteses [Pidd 1994, Chwif and Medina 2006]. Além disso, simulações possuem teor profilático, pois possibilitam predizer o desempenho e identificar gargalos e déficits antes de construir ou modificar um sistema, poupando recursos e investimentos [Carson 2004, Chwif and Medina 2006].

2.2. Simulação Distribuída

Uma simulação é dita sequencial ou centralizada quando sua execução é conduzida em uma máquina com uma única unidade de processamento. Em contrapartida, os campos de simulação paralela e distribuída exploram a descentralização dos modelos. Em geral, simulação paralela enfatiza a decomposição do modelo entre as múltiplas CPUs de uma única máquina ou sistema computacional fortemente acoplado. Já a simulação distribuída preocupa-se em particionar o modelo em máquinas e plataformas computacionais geograficamente espalhadas, autônomas, heterogêneas e interconectadas via rede [Fujimoto 2000, Fujimoto 2003, Fujimoto 2015]. Visto que as duas áreas, embora distintas, compartilham problemas e desafios, no corrente trabalho elas serão unidas sob o termo "simulação distribuída" (DS) [Fujimoto 2015].

Em DS, cada subdivisão do modelo de simulação constitui um Processo Lógico (PL), também chamado de simulador. A comunicação entre os PLs dá-se pela troca de mensagens, em que cada uma representa um evento enviado de um simulador a outro. Em uma simulação de uma rede aeroportuária, por exemplo, um evento pode simbolizar a viagem de um avião de um aeroporto a outro. Cada mensagem carrega um carimbo temporal (*timestamp*), o qual informa o instante de início do evento no tempo simulado [Fujimoto 2000, Fujimoto 2003].

A toda simulação é imposta uma limitação conhecida como *Local Causality Constraint* (LCC), a qual estipula que todos os PLs devem processar os eventos em ordem de estampa temporal. Para uma simulação distribuída, está comprovado que, caso a LCC seja respeitada, a execução da simulação produzirá resultados idênticos aos de uma execução centralizada do mesmo modelo. Se não ocorrer o cumprimento da LCC, o processamento de um evento pode afetar o passado de outro, comprometendo a acurácia da simulação. Uma transgressão da LCC é denominada "violação de causalidade", e surge quando um PL recebe uma mensagem com carimbo menor que o seu relógio local (LVT) [Fujimoto 2000, Fujimoto 2003].

A aderência à LCC a fim de evitar as violações temporais supracitadas não é algo trivial de ser implementado em DS. Isso, pois um PL não sabe, *a priori*, quais eventos serão agendados para ele por outros PLs. Não há como prever se um PL receberá um evento com estampa menor que a do seu relógio local. Esse obstáculo, conhecido como "problema da sincronização" ou "problema do gerenciamento de tempo", é um tema recorrente em M&S e estimulou a elaboração de mecanismos para solucioná-lo [Fujimoto 2000, Fujimoto 2003].

2.3. Algoritmos de Sincronização

Algoritmos de sincronização (ou algoritmos de gerenciamento de tempo) foram desenvolvidos para garantir que simulações distribuídas cumprissem a LCC. De modo geral, esses algoritmos costumam ser separados em duas categorias: conservadores (ou síncronos) e otimistas (ou assíncronos) [Fujimoto 2000, Fujimoto 2003, Lindén 2018].

No protocolo conservador, concebido pelos trabalhos de [Chandy and Misra 1979] e [Bryant 1977], a principal incumbência desempenhada é determinar quando é seguro processar um dado evento. Um evento é dito "seguro" quando é certo que o PL em questão não receberá, no futuro, eventos com estampa temporal menor que a desse evento. Essa garantia é obtida através de um valor de LBTS (*Lower Bound on*

the Time Stamp), que permite ao algoritmo decidir quais eventos são seguros. Somente eventos com carimbo menor que LBTS são considerados seguros e, assim, podem ser processados. Dessarte, técnicas conservadoras previnem por completo a ocorrência de violações de causalidade na simulação [Fujimoto 2000, Fujimoto 2003, Fujimoto 2015].

Protocolos otimistas principiaram-se com o algoritmo Time Warp, desenvolvido por [Jefferson 1985]. De modo geral, métodos assíncronos permitem que violações temporais aconteçam, mas são capazes de detectá-las e recuperar-se delas. Time Warp é composto por dois mecanismos complementares: local e global. O mecanismo local autoriza o livre processamento de eventos por parte dos PLs. Contudo, se ocorrer uma violação de causalidade, é acionado um procedimento de *rollback*, que reverte as mudanças realizadas até a estampa de tempo da violação e, em seguida, reprocessa os eventos em ordem de carimbo até o instante do relógio local do PL. O mecanismo global, por sua vez, introduz o conceito do GVT (*Global Virtual Time*), que age como um limite inferior no carimbo de *rollbacks* futuros, de modo que *checkpoints* antecedentes ao GVT podem ser liberados da memória [Fujimoto 2003, Fujimoto 2015, Jefferson 1985].

2.4. Sincronização Híbrida

As principais vantagens do algoritmo síncrono são a sua facilidade de entendimento e implementação. Outrossim, o algoritmo assíncrono destaca-se por promover um ganho de desempenho para uma ampla gama de modelos. Entretanto, embora os protocolos também garantam o corretismo da simulação, sua adoção global — em todos os PLs — pode trazer prejuízos. Uma simulação inteiramente conservadora depende de um valor adequado de *lookahead* e pode estar sujeita a lentidão, ociosidades e impasses. Já uma simulação totalmente otimista é mais complexa de efetivar e pode sofrer de violações de causalidade e consequente excesso de *rollbacks* [Parizotto and Mello 2022, Jefferson and Barnes 2017].

Levando isso em consideração, uma abordagem promissora — com o nome de "sincronização híbrida" — consiste em combinar simuladores síncronos e assíncronos em uma mesma simulação. Existem várias técnicas para empreender uma sincronização híbrida, como:

1. a simulação inteira alterna entre conservadora e otimista;
2. PLs alternam entre conservador e otimista, individualmente;
3. a simulação compõe-se de PLs conservadores e otimistas, sem alternância individual de protocolo.

Independentemente da estratégia, o objetivo é tirar proveito dos benefícios dos dois protocolos e superar os detrimentos intrínsecos a cada um [Parizotto and Mello 2022, Jefferson and Barnes 2017, Eker et al. 2021].

Não obstante, conciliar PLs com primitivas de sincronização opostas não é algo trivial de se implementar. Um PL síncrono recebendo mensagens de um assíncrono deve desconsiderar mensagens que causariam uma violação temporal. Em compensação, um PL assíncrono pode executar um *rollback* e descartar mensagens provenientes de um PL síncrono. Em razão disso, conclui-se que uma sincronização híbrida superficial e desregrada é propensa a deteriorar a condução e acurácia da simulação [Parizotto and Mello 2022].

3. Trabalhos correlatos e motivação

Nesta seção são expostos e discutidos alguns estudos relacionados aos objetivos desta monografia. Comumente a vários trabalhos envolvendo escalonamento de mensagens em simulação computacional, encontra-se a política de agendamento LTF (*Lowest Timestamp First*). É um algoritmo simples e difundido em M&S que determina que o próximo evento a ser agendado deve ser o evento com a menor estampa temporal [Som and Sargent 1998, Santoro and Quaglia 2010]. De modo geral, as pesquisas subsequentes abordam a LTF, ou para experimentá-la em simulações ou para propor soluções diferentes em virtude de déficits encontrados nessa política.

Dentre os correlatos identificados nesta seção, o estudo apresentado em [Parizotto and Mello 2022] é a principal solução referenciada, e foi alicerce para a contribuição apresentada no presente trabalho. Nele, a sincronização híbrida foi tratada como um problema de escalonamento. Um algoritmo escalonador de mensagens foi proposto, norteado por duas métricas: soma de trabalho exercido e total de violações causais. O intuito do algoritmo é selecionar as mensagens que mais contribuem para a acurácia da simulação. Por se tratar do principal trabalho correlato, uma discussão mais cuidadosa de detalhes da estratégia de escalonamento é apresentada na próxima seção. Na mesma oportunidade, faz-se a argumentação e localização das novas contribuições do presente trabalho.

O artigo de [Som and Sargent 1998] abordou a utilidade da diretiva LTF, sob o seguinte ponto de vista: o evento mais próximo tem, em tese, a menor chance de ser alvo de um *rollback* futuro. Visando aprimorar esse cenário, os autores conceberam um algoritmo de escalonamento que estima explicitamente a probabilidade de um evento ser revertido por outro evento atrasado (*straggler*). A técnica apresentada difere-se de LTF, pois seleciona o próximo evento com base na chance da ocorrência de um retrocesso futuro. Experimentos em um simulador atestaram que o escalonador probabilístico, sob certas condições, reduziu substancialmente o número de *rollbacks*, aumentando a eficácia da simulação. Comparando com as direções do presente trabalho, o algoritmo de [Som and Sargent 1998] considera apenas o protocolo otimista, e não opera com eventos que contenham um custo de tempo.

Em [Xiao et al. 1999], foram enumeradas duas estratégias para escalonamento de PLs/eventos em sistemas multiprocessadores: usar uma fila centralizada de eventos e dividir os eventos entre as unidades. Ambas possuem problemas, como competição de acesso à fila, baixa localidade de *cache* e desbalanceamento de carga. Os autores revisitaram a abordagem da fila compartilhada, aprimorando-a com um método de escalonamento em três âmbitos: grupos de PLs, PLs e eventos. O objetivo do algoritmo CCT (*Critical Channel Traversing*) é agendar o PL com mais eventos prontos para execução e que, portanto, realizará mais trabalho em uma mesma sessão. O teste de validação conduzido apontou que CCT proporcionou um ganho expressivo no número de eventos processados por segundo. No entanto, o escalonador CCT trabalha apenas com a primitiva conservadora e não leva em conta eventos com carga de tempo.

O trabalho de [Liu et al. 2001], assim como [Xiao et al. 1999], focou em configurações multiprocessadores. Para determinar a janela de tempo segura de um PL e garantir um rápido escalonamento, o escalonador precisa acessar variáveis compartilhadas por múltiplos processadores. Uma solução usando bloqueios (*locks*) para exclusão

mútua traz desvantagens, como gargalos de desempenho, atrasos na consecução do acesso e situações de impasse. Os autores propuseram um escalonador sem bloqueios (*lock-free*) de PLs que minimiza as sobrecargas necessárias para computar a janela de tempo segura de cada PL. Foi produzido um experimento cotejando com o escalonador CCT de [Xiao et al. 1999]. Os resultados sugeriram que, no geral, o algoritmo sem bloqueios auferiu uma redução considerável no tempo de execução, nos casos em que havia poucos PLs por processador. À semelhança de [Xiao et al. 1999], o escalonador foi projetado para simulações síncronas e desconsidera mensagens com custo temporal.

Em [Santoro and Quaglia 2010], discutiu-se o escalonamento de PLs com base na política LTF, assim como [Som and Sargent 1998]. Conquanto simples e correta, LTF é onerosa quando os eventos estão distribuídos entre vários PLs. Pensando nisso, foi proposto um escalonador de PLs pautado em LTF e operando em tempo constante e com pouca sobrecarga (*overhead*). Um teste em uma plataforma de simulação distribuída verificou uma redução progressiva do tempo de execução da simulação, paragonando com um escalonador LTF de complexidade linear. Confrontando com as metas do corrente trabalho, o escalonador de [Santoro and Quaglia 2010] foi implementado para simulações otimistas, não considera eventos que carreguem uma duração, ou tempo de processamento, e baseia-se unicamente na diretiva LTF.

4. Modelo de escalonamento de mensagens

Esta seção discorre, inicialmente, sobre a solução proposta na principal referência correlata, ou referência base, deste trabalho e expõe os conceitos e procedimentos envolvidos no escalonamento de mensagens. Em seguida, os ajustes e contribuições acrescentados ao algoritmo de escalonamento original são detalhados ao tempo em que a especificação completa do algoritmo é discutida e explicada. O histórico de estudos de que este trabalho deriva tem usado o DCB (*Distributed Co-simulation Backbone*) como plataforma para experimentação. DCB é um representativo exemplo de simulador distribuído que aborda desafios atuais em sincronização híbrida, representada em sua arquitetura. Assim, por terceiro, esta seção relata a etapa de integração da solução ao DCB, antecedida de uma breve descrição da plataforma.

4.1. Apresentação da solução base

Conforme explicado anteriormente, sincronização híbrida é uma técnica promissora, porém carregada de desafios quanto à sua efetivação. Em [Parizotto and Mello 2022] essa técnica foi discutida com foco nos conjuntos de mensagens processadas pela simulação. Os autores analisaram a fila de entrada do PL, composta de mensagens com estampas temporais de início, conforme tempo local do PL. Além da estampa temporal, cada mensagem possui também um valor de "trabalho" ou "carga", correspondente à duração, em unidades de tempo simulado, do evento. Ou seja, ao receber uma mensagem, o PL destino executa um evento com tempo de início (estampa temporal) e tempo de execução (trabalho).

Cada PL processa uma única mensagem por vez. Supondo ser m_i a mensagem vigente, o simulador inicia seu processamento no tempo $t(m_i)$ — carimbo de início de m_i — e finaliza em $t(m_i) + w(m_i)$ — ou seja, o carimbo inicial somado ao trabalho de m_i. Dependendo da distribuição de mensagens na fila de entrada, podem existir conflitos,

isto é, mensagens cujos intervalos sobrepõem-se. Nesses casos, uma estratégia comum é retirar uma fração das mensagens, segundo algum critério, até que não haja mais conflitos no conjunto e o tratamento da fila de entrada possa ser retomado. Quanto mais mensagens forem descartadas, maior é a perda de precisão da simulação, visto que são eventos que não puderam ser processados [Parizotto and Mello 2022].

Em [Parizotto and Mello 2022] foram identificadas, para uma coleção de mensagens qualquer, duas métricas pertinentes. A primeira é o número de violações temporais geradas pelo processamento ordinário das mensagens. A segunda é o total de trabalho exercido, ou seja, o somatório das durações dos eventos executados com sucesso. Na visão dos autores, uma simulação com sincronização híbrida poderia ser otimizada com base nessas duas métricas. Todavia, para determinados arranjos de mensagens, pode ser difícil ou mesmo impossível minimizar a quantidade de violações e, concomitantemente, maximizar a soma de trabalho.

Visando mitigar essa dicotomia, [Parizotto and Mello 2022] propuseram um algoritmo de escalonamento que atua como um filtro sobre a fila de entrada do PL. Baseando-se na premissa de que, quanto mais mensagens forem agendadas, menor será o total de violações temporais, o algoritmo seleciona um subconjunto de mensagens de modo que haja um equilíbrio entre quantidade de mensagens e somatório de trabalho. A coleção "ideal" de mensagens, que maximizaria ambas as métricas, é denominada "ponto utópico", e o subconjunto escolhido é o que mais se aproxima desse ponto.

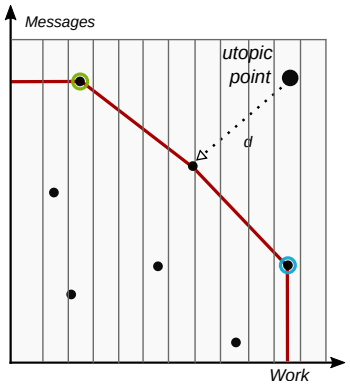


Figura 1. Subconjuntos e o ponto utópico (fonte: [Parizotto and Mello 2022])

A Figura 1 exibe um plano cartesiano, em que cada ponto representa um subconjunto de mensagens. O eixo X denota a soma de trabalho, e o eixo Y a quantidade de mensagens. Pode-se observar que o ponto verde otimiza apenas o total de mensagens, enquanto o ponto azul otimiza somente o total de trabalho. O ponto utópico, por sua vez, representa o processamento de todas as mensagens disponíveis, algo que não é factível em um cenário com conflitos. O emprego do algoritmo escalonador possibilitaria selecionar o subconjunto com a menor distância até o ponto utópico — na Figura 1, é o ponto indicado pela seta pontilhada.

4.2. Escalonamento de mensagens

Esta subseção inicia com uma explanação de conceitos relativos ao processo de escalonamento de mensagens. Na sequência, são introduzidas as contribuições efetuadas no

algoritmo escalonador original, seguidas da exposição e descrição da nova versão algoritmo que agrega as contribuições do presente trabalho.

4.2.1. Primitivas de escalonamento

Como mencionado, o algoritmo opera sobre as filas de entrada dos PLs. Uma fila $Q = \{m_1, m_2, m_3, \ldots\}$ contém as mensagens ainda não processadas, ordenadas de modo crescente pelas *timestamps*. Cada mensagem $m_i \in Q$ possui uma estampa temporal de início $t(m_i)$, um valor de trabalho $w(m_i)$ e uma estampa de fim $f(m_i) = t(m_i) + w(m_i)$. Interceptando a fila de entrada, o algoritmo utiliza programação dinâmica para escolher o subconjunto $S \subseteq Q$ de mensagens que atinja um equilíbrio entre número de mensagens e soma de trabalho [Parizotto and Mello 2022].

O algoritmo utiliza-se de dois componentes internos para calcular o escalonamento: a função $p()$ e a função $utp()$. Ambas serão examinadas na sequência.

4.2.2. A função $p()$

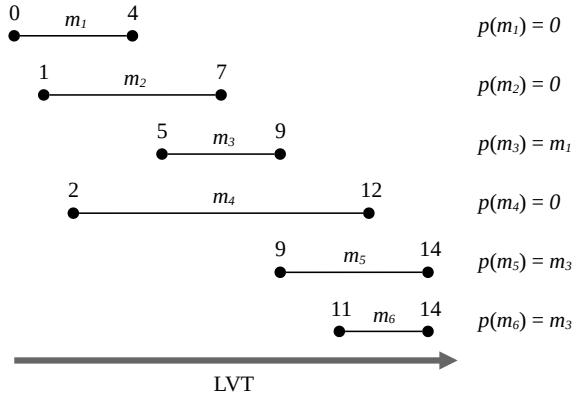


Figura 2. Exemplo da função $p()$

Dado um conjunto C de mensagens ordenadas crescentemente pelo carimbo temporal de fim, a função $p()$ recebe como parâmetro uma mensagem $m \in C$ e encontra a mensagem $n \in C$ anterior mais próxima tal que m e n sejam compatíveis (i.e., não possuam sobreposição) [Kleinberg and Tardos 2006]. Na Figura 2, por exemplo, considerando o conjunto $C = \{m_1, m_2, m_3, m_4, m_5, m_6\}$, verifica-se que $p(m_5) = m_3$, uma vez que m_4 — a mensagem imediatamente anterior — possui sobreposição com m_5. Por outro lado, tanto m_1 quanto m_2 não possuem uma mensagem prévia compatível, de modo que o $p()$ dessas mensagens é nulo (ou, conforme convencionado na Figura 2, é computado como 0).

Com o objetivo de reduzir o custo dos cálculos de $p()$ no decorrer do escalonamento, optou-se por memoizar a função $p()$. Isto é, os resultados dos cômputos de $p()$ são armazenados para futuras consultas, reduzindo os recálculos necessários. No entanto, levando em conta o contexto de simulação distribuída e o fato de a coleção de mensagens ser alterada em razão do processamento (retirada) e recebimento (acréscimo) de mensagens, foi necessário estipular uma política para atualizar essa memoização, onde:

- Seja C o conjunto de mensagens na fila de entrada
- Seja n a nova mensagem recebida ou a mensagem retirada do conjunto
- Para toda mensagem $m \in C$, é aplicado o seguinte critério:
  - Se $t(m) \geq f(n)$, então $p(m)$ precisa ser recalculado
  - Senão, o valor armazenado (memoizado) ainda é válido e pode ser aproveitado

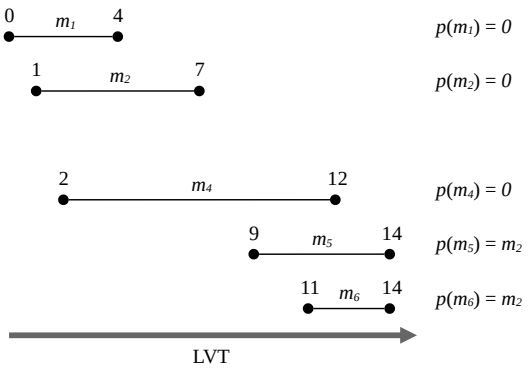


Figura 3. Memoização de $p()$: cenário anterior

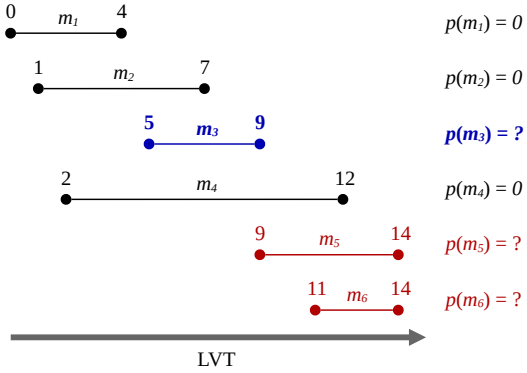


Figura 4. Memoização de $p()$: cenário posterior

A Figura 3 representa um cenário anterior à chegada de m_3, e a Figura 4 ilustra a mudança suscitada pelo recebimento da nova mensagem. Mesmo com esse acréscimo, o $p()$ preexistente de m_1, m_2 e m_4 (calculado em uma iteração prévia do algoritmo de escalonamento), pode ser aproveitado. Para m_5 e m_6, todavia, o valor memoizado torna-se obsoleto e necessita ser recomputado antes de ser utilizado novamente. Além disso, o $p()$ da nova mensagem — $m3$ — deve ser calculado, visto que ainda não existe.

4.2.3. A função $utp()$

A segunda subfunção, $utp()$, é responsável por calcular o ponto utópico supracitado, o qual é usado durante o escalonamento como parâmetro para a escolha do subconjunto de mensagens [Parizotto and Mello 2022]. Supondo um conjunto C de mensagens ordenadas crescentemente pela estampa temporal de fim, para uma dada mensagem $m_i \in C$ é definido o subconjunto $C_i \subseteq C$. C_i é composto por m_i mais as mensagens que aparecem antes de m_i na fila de entrada, i.e., $C_i = \{m_j \in C \mid j \leq i\}$. Em cima desse conjunto, é calculado o $utp(C_i)$ — ponto utópico do conjunto. Esse ponto é composto por dois valores: a soma dos trabalhos das mensagens do conjunto e a cardinalidade do conjunto. Para facilitar, será usada também a notação $utp(m_i)$, de modo que $utp(m_i) = utp(C_i)$.

A Figura 5 utiliza o mesmo conjunto de mensagens da Figura 2, com a inclusão do trabalho (w) de cada mensagem para auxiliar na visualização. A ordem do conjunto de mensagens é $C = \{m_1, m_2, m_3, m_4, m_5, m_6\}$, uma vez que essa ordenação é feita pelo instante de fim. Assim, para a mensagem m_2, que está na segunda posição, o seu $utp()$ consiste em: soma do seu trabalho com os das mensagens precedentes (i.e., $w(m_1) + w(m_2) = 10$), e número de mensagens desde o início do conjunto até m_2, inclusive (2).

Assim como a função $p()$, a função $utp()$ também pode ser memoizada, mas com ressalvas devido às remoções e acréscimos de mensagens ao conjunto. O critério para

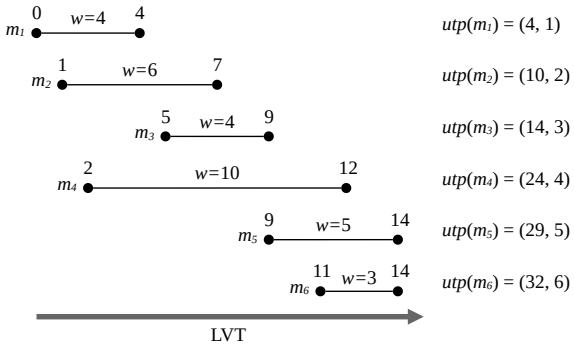


Figura 5. Exemplo da função $utp()$

aproveitamento ou descarte dos valores armazenados na memoização está disposto na sequência:

- Seja C o conjunto de mensagens na fila de entrada, ordenadas pela estampa temporal de fim
- Seja m_j a nova mensagem recebida ou a mensagem retirada do conjunto
- Para toda mensagem $m_i \in C$, é aplicado o seguinte critério:
  - Se $i < j$, ou seja, se m_i está posicionada antes de m_j na fila de entrada, então o $utp(m_i)$ preexistente pode ser aproveitado
  - Senão, precisa ser recalculado

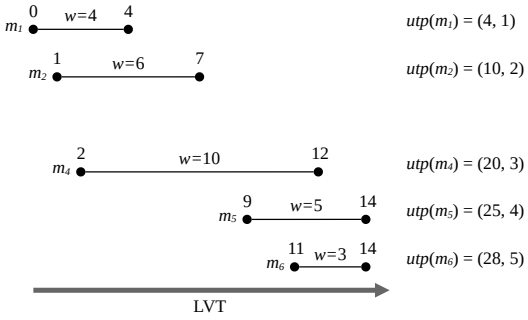


Figura 6. Memoização de $utp()$: cenário anterior

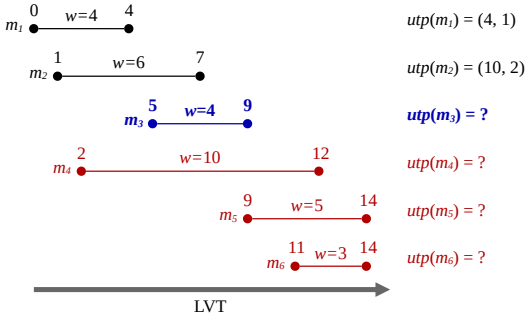


Figura 7. Memoização de $utp()$: cenário posterior

A Figura 6 constitui uma situação em que m_3 ainda não foi incluída e, portanto, não é contabilizada nos pontos utópicos já memoizados. Após o recebimento dessa mensagem e sua inserção ordenada no conjunto, na Figura 7, é necessário descartar o $utp()$ das mensagens que aparecem depois de m_3, a saber: m_4, m_5 e m_6. Vale salientar que $utp(m_3)$, por não existir, também deve ser computado e armazenado.

4.2.4. Algoritmo de escalonamento

O Algoritmo 1 contém o algoritmo de escalonamento em sua totalidade, o qual emprega as funções $p()$ e $utp()$ suprarreferidas. O algoritmo funciona com base em programação dinâmica, de modo que o escalonamento de uma mensagem é derivado dos escalonamentos das mensagens anteriores a ela [Parizotto and Mello 2022,

Dados: M: fila de entrada ordenada; V: conjunto de mensagens já visitadas; $memoS$: tabela de memoização para os escalonamentos; $memoP$: tabela de memoização para a função $p()$; $memoUTP$: tabela de memoização para a função $utp()$

Função `Receber(`k`)`**:**
  $M \leftarrow M \cup \{k\}$
  $V \leftarrow \{a \in M \mid a < k\}$
  para cada $j \in M$ **faça**
    se $(j = k) \parallel (t(j) \geq f(k))$ **então**
      $memoP[j] \leftarrow p(j)$
    se $j \geq k$ **então**
      $memoUTP[j] \leftarrow utp(j)$
  fim
  $m \leftarrow$ última mensagem de M
  $memoS[m] \leftarrow$ `Escalonar(`m`)`

Função `Escalonar(`m`)`**:**
  se $(m = 0) \parallel (m \in V)$ **então**
    retorna $memoS[m]$
  $utp \leftarrow memoUTP[m]$
  $n \leftarrow memoP[m]$
  $memoS[n] \leftarrow$ `Escalonar(`n`)`
  $S \leftarrow memoS[n] \cup \{m\}$
  $V \leftarrow V \cup \{n\}$
  $k \leftarrow$ mensagem imediatamente anterior a m
  $memoS[k] \leftarrow$ `Escalonar(`k`)`
  $T \leftarrow memoS[k]$
  $V \leftarrow V \cup \{k\}$
  retorna $min($`Custo(`*S, utp*`)`, `Custo(`*T, utp*`)`$)$

Função `Custo(`C, utp`)`**:**
  retorna $\sqrt{(C.work - utp.work)^2 + (C.msg - utp.msg)^2}$

Algoritmo 1: Algoritmo escalonador de mensagens

Kleinberg and Tardos 2006]. Por isso, é utilizada uma estrutura para memoizar os resultados dos escalonamentos e aproveitá-los em execuções futuras do algoritmo [Parizotto and Mello 2022].

No preâmbulo do algoritmo, são declaradas algumas estruturas de armazenamento, como os conjuntos M (que representa a fila de entrada) e V (explicado mais adiante). Constam também as tabelas para guardar as memoizações de $p()$ e $utp()$, que configuram adições trazidas por este trabalho. Por outro lado, a estrutura de memoização dos escalonamentos já havia sido mencionada em [Parizotto and Mello 2022]; aqui, apenas foi feita a sua declaração explícita.

Ao receber uma mensagem k, considera-se que as mensagens anteriores a k já foram "visitadas", i.e., seus escalonamentos armazenados ainda são válidos e podem

ser aproveitados (linha 3). Na sequência, são aplicados os critérios de atualização das memoizações de $p()$ e $utp()$ (linha 4), conforme já explicado, que constituem uma das contribuições deste trabalho. A última mensagem na fila é usada para calcular o escalonamento da fila em si, já que o seu escalonamento deriva de todas as outras mensagens. Esse valor também é memoizado (linha 11).

O caso base do escalonamento (linha 13) ocorre quando a mensagem m foi marcada como visitada, quando não possui $p()$ ou quando não possui uma mensagem anterior (a notação adotada para esses últimos dois casos é $m = 0$). Após, o algoritmo calcula dois subconjuntos candidatos para o escalonamento de m. O primeiro é composto por m mais o escalonamento de $p(m)$ (linha 18). O segundo equivale ao escalonamento da mensagem que precede m na fila de entrada (linha 22). Os ajustes efetuados pelo presente trabalho consistem na utilização das tabelas de memoização, tanto para consultar o valor de $p()$ quanto para armazenar os escalonamentos internos computados.

Para decidir entre os dois subconjuntos, é empregada a função de custo (linha 25). Essa função recebe dois conjuntos — o subconjunto candidato e o ponto utópico — e calcula a distância euclidiana entre eles, com base nas métricas de soma de trabalho e cardinalidade do conjunto. Assim, o subconjunto escolhido é o que possui a menor distância até o ponto utópico (linha 24). Vale ressaltar que o ponto utópico passou a ser obtido através de uma consulta à estrutura de memoização correspondente.

4.3. Integração à plataforma de simulação

Embasando-se nas decisões tomadas em [Parizotto and Mello 2022], a arquitetura DCB (*Distributed Co-simulation Backbone*) foi escolhida para a incorporação e experimentação do algoritmo. DCB é um ambiente de simulação distribuída que objetiva oferecer um mecanismo genérico para a cooperação entre Processos Lógicos heterogêneos. Trata-se de um sistema flexível, que permite integrar simuladores com o mínimo de reprogramação necessária. Além disso, a especificação de DCB já prevê suporte a sincronização híbrida [Mello and Wagner 2002].

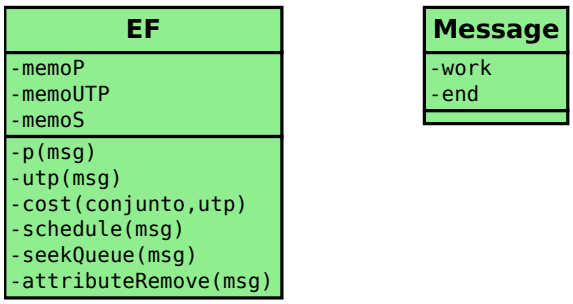


Figura 8. Diagrama parcial da implementação de DCB

A plataforma é escrita na linguagem de programação Java; sendo assim, o algoritmo foi codificado na mesma linguagem. A Figura 8 exibe um diagrama de classes parcial da arquitetura DCB, contendo somente os módulos que foram modificados. Na classe EF, que representa um Processo Lógico, foram incluídas as estruturas de dados para as memoizações de $p()$, $utp()$ e escalonamento. O algoritmo escalonador foi implementado no método $schedule()$, e suas funções internas foram deixadas em métodos separados, invocados dentro do método principal. O método preexistente $seekQueue()$,

que adiciona e despacha as mensagens na fila de entrada, foi ajustado para acomodar a função $Receber()$ do Algoritmo 1, e o método $attributeRemove()$, responsável por remover mensagens da fila, foi reescrito para aplicar os critérios aludidos de atualização das memoizações. Por último, na classe $Message$, foram anexados dois atributos: trabalho da mensagem e estampa de fim (o atributo de estampa inicial já existia).

A incorporação supracitada do algoritmo de escalonamento forneceu o suporte necessário para complementar os testes efetuados no artigo-base. A validação numérica apresentada em [Parizotto and Mello 2022] é importante, contudo não substitui a experimentação em um sistema efetivo de simulação. Com a operacionalização do escalonamento proposto, foi possível executar modelos na arquitetura DCB e observar seus resultados, conforme está descrito na próxima seção.

5. Experimentos e análises

Nesta seção, primeiramente, são apresentados os modelos simulados na arquitetura DCB, seus parâmetros e variações, e as métricas coletadas a partir das execuções. Em seguida, os resultados das simulações são exibidos e discutidos.

5.1. Parâmetros, configurações e métricas

Foram selecionados dois parâmetros para serem variados na construção dos modelos de simulação. O primeiro trata-se da taxa de envio de mensagens, que consiste na quantidade de mensagens enviadas por cada PL entre cada avanço do seu LVT. O segundo é o total de tempo simulado — também chamado de "tamanho" da simulação.

Modelo	Taxa de envio	Tempo total
1	10	2000
2	20	2000
3	30	2000
4	40	2000
5	50	2000
6	10	1000
7	10	3000
8	10	4000
9	10	5000

Tabela 1. Modelos de simulação

Foram definidos, ao todo, nove configurações de modelos, conforme constante na Tabela 1. Os modelos 1–5 mantêm o tempo total fixo enquanto variam linearmente a taxa de envio de mensagens, de 10 a 50. O intuito aqui é verificar o impacto do aumento da taxa de mensagens sobre o número de mensagens agendadas e soma de trabalho processado.

Os modelos 6, 1, 7, 8 e 9 — nessa ordem — fixam a taxa de envio de mensagens enquanto variam linearmente o tempo simulado total, de 1000 a 5000. Assim, pode-se observar o efeito da variação do tempo total sobre os resultados. Vale frisar que o modelo 1 é usado para ambas as análises.

Quatro métricas foram coletadas das simulações: mensagens processadas, mensagens descartadas, trabalho processado e trabalho descartado. Esses indicadores condizem

com as respectivas medidas empregadas pelo algoritmo de escalonamento. Para mitigar possíveis resultados anômalos, cada modelo foi executado cinco vezes, e as métricas das cinco replicações foram agregadas via média aritmética.

Cada modelo também foi rodado tanto com o algoritmo de escalonamento proposto quanto com a diretiva LTF. Isso possibilitou comparar o desempenho da solução proposta com o da política LTF convencional e avaliar o diferencial trazido pelo novo algoritmo. Ademais, foram conduzidas simulações tanto em um cenário local — com uma única máquina hospedando o modelo inteiro — quanto distribuído — no qual o modelo foi igualmente particionado em duas máquinas.

É importante salientar que, no caso dos modelos distribuídos supracitados, buscou-se amenizar aspectos derivados da distribuição em si, como latência de rede e mensagens em trânsito, de modo a não comprometer as métricas finais desejadas. Também faz-se necessário apontar que os modelos estipulados na Tabela 1 foram projetados para "estressar" as políticas de escalonamento, sujeitando-as a situações com circulação massiva de mensagens conflitantes. O propósito foi testar e avaliar a resiliência das diretivas, e não necessariamente reproduzir comportamentos encontrados em modelos reais de simulação.

5.2. Variando a taxa de envio de mensagens

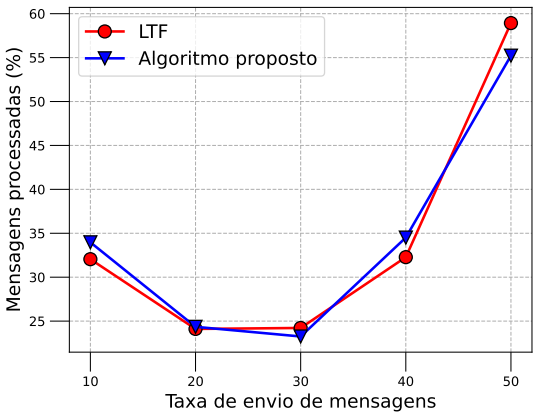


Figura 9. Modelos locais, taxa de envio variável e mensagens processadas

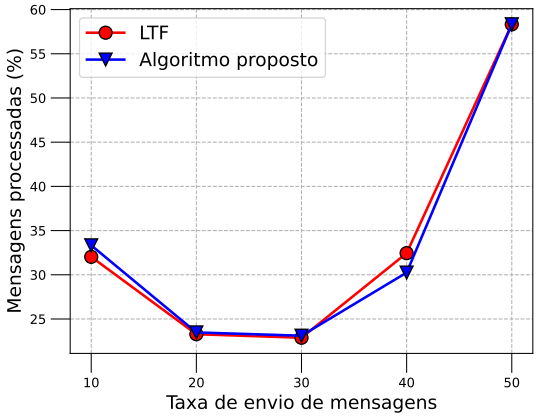


Figura 10. Modelos distribuídos, taxa de envio variável e mensagens processadas

Na Figura 9, percebe-se que, no geral, tanto o algoritmo proposto quanto a política LTF processam uma porcentagem similar de mensagens. Ao variar a taxa de envio de mensagens, ocorre uma alternância mínima entre as duas diretivas quanto ao maior escore. No cenário distribuído da Figura 10, os resultados assemelham-se aos da Figura 9, exceto que o algoritmo proposto tende a se aproximar de LTF em taxas maiores.

Na Figura 11, para a métrica de trabalho, o algoritmo proposto apresenta vantagem em taxas menores. Aumentando a quantidade de mensagens, os desempenhos de ambas as diretivas convergem. Nas simulações distribuídas da Figura 12, o comportamento mantém-se, exceto que as performances das políticas tornam-se paralelas em taxas maiores.

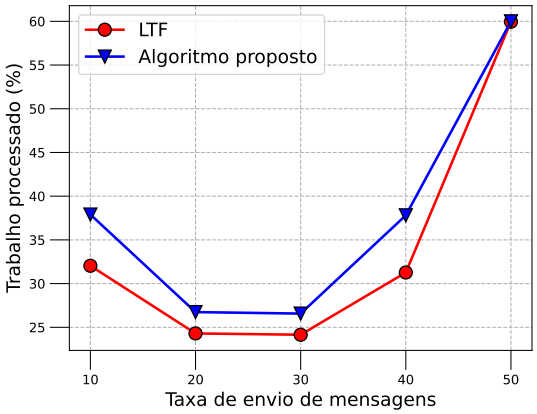


Figura 11. Modelos locais, taxa de envio variável e trabalho processado

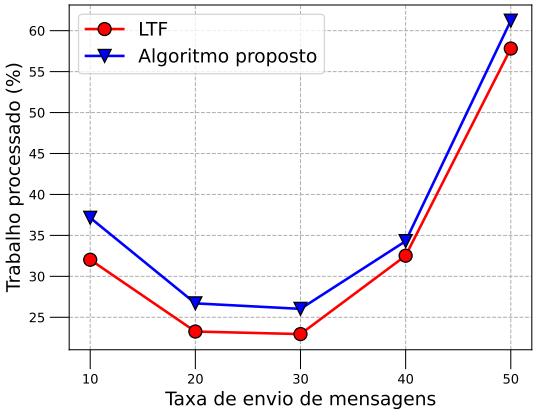


Figura 12. Modelos distribuídos, taxa de envio variável e trabalho processado

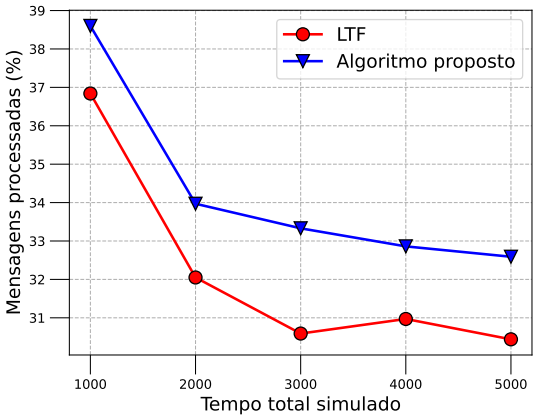


Figura 13. Modelos locais, tempo total variável e mensagens processadas

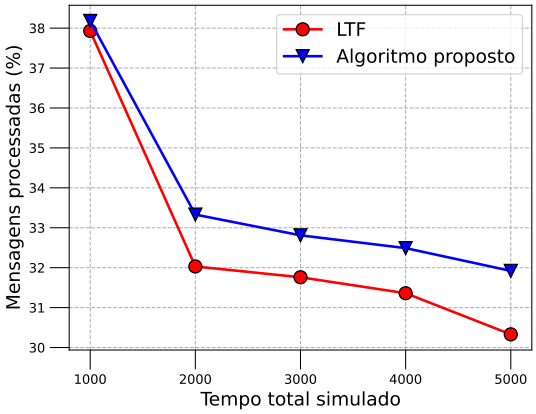


Figura 14. Modelos distribuídos, tempo total variável e mensagens processadas

5.3. Variando o tempo total simulado

Alterando o tempo total, nas Figuras 13 e 14, o algoritmo proposto demonstra vantagem maior em relação à política LTF, comparando com as simulações supracitadas que variaram a taxa de envio. Em ambas as figuras, pode-se observar que o algoritmo processa mais mensagens que LTF, e o aumento do tamanho da simulação não afetou o desempenho do algoritmo proposto. Nas execuções distribuídas, a performance de LTF aproxima-se mais da do algoritmo, sem, no entanto, superá-la.

Referente à métrica de trabalho processado, as Figuras 15 e 16 exibem um comportamento similar às outras figuras que variaram o tempo total simulado. Das simulações locais para as distribuídas, o desempenho também foi semelhante. Todavia, percebe-se que, em termos percentuais, o algoritmo processa mais trabalho do que mensagens. Isso indica que a solução proposta é mais eficaz na otimização da métrica de soma de trabalho.

Analisando todos os resultados exibidos na seção corrente, constata-se que há uma porcentagem baixa de mensagens processadas e trabalho processado. Esse comportamento já era esperado, pois, conforme afirmado anteriormente, o objetivo dos experi-

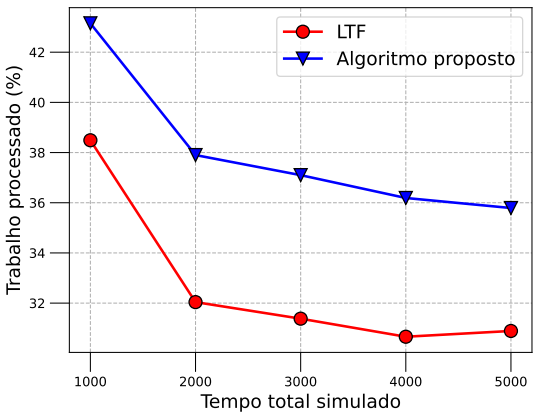


Figura 15. Modelos locais, tempo total variável e trabalho processado

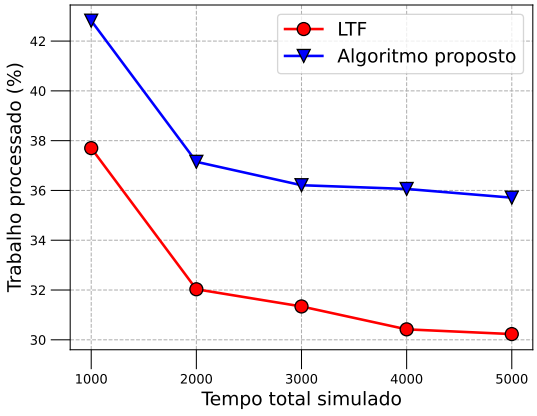


Figura 16. Modelos distribuídos, tempo total variável e trabalho processado

mentos foi gerar volumes numerosos de mensagens sobrepostas. Isso amplifica a fração de mensagens descartadas — e, consequentemente, o trabalho descartado.

Os experimentos também mostraram que o algoritmo proposto cumpriu sua função de buscar o equilíbrio entre as duas métricas já mencionadas, especialmente nos testes com variação no tamanho da simulação. Portanto, é possível argumentar que, cotejando com a política LTF, o escalonamento proposto aproximou-se mais do ponto utópico, ajudando a atenuar a dicotomia entre total de mensagens processadas e soma de trabalho exercido.

5.4. Desativando as memoizações

Visando testar o impacto das memoizações propostas de $p()$ e $utp()$, também foram conduzidos experimentos com e sem essas memoizações. Foi escolhido somente o modelo 9 da Tabela 1 para ser simulado. Quanto às memoizações, foram definidas as seguintes variações: somente a memoização de $p()$, somente a memoização de $utp()$ e com ambas desligadas. Cada variação foi executada cinco vezes, de modo centralizado, e as métricas das cinco replicações foram agregadas via média aritmética. Ademais, na comparação foram incluídos os resultados do modelo 9 com todas as memoizações ativas, já coletados.

As Figuras 17 e 18 exibem os resultados para as variações aludidas, no que diz respeito ao total de mensagens processadas e soma de trabalho processado, respectivamente. Percebe-se que, em ambos os casos, remover a memoização de $utp()$ traz mais prejuízo do que remover a memoização de $p()$. Além disso, nota-se que desligar todas as memoizações causa uma redução tanto em mensagens quanto em trabalho processado, confrontando com a variação em que ambas estão ativadas. Para simulações com maior tamanho ou maior número de mensagens, essa diferença poderia escalar e amplificar-se, causando um impacto ainda maior na acurácia.

A despeito dos resultados das Figuras 17 e 18, argumenta-se que outros testes, com mais relevância para a análise, poderiam ser efetuados. Em especial, mais modelos poderiam ser executados, tanto da Tabela 1 quanto de outras configurações, para sondar se a tendência trazida pelas memoizações permanece. Além disso, outras métricas diferentes de mensagens e trabalho processados podem ser consideradas, como tempo físico

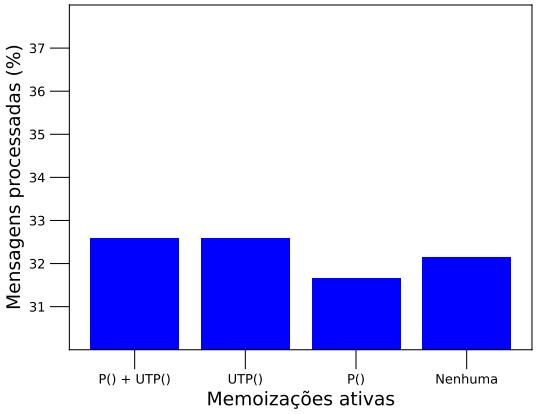


Figura 17. Mensagens processadas com memoizações desativadas

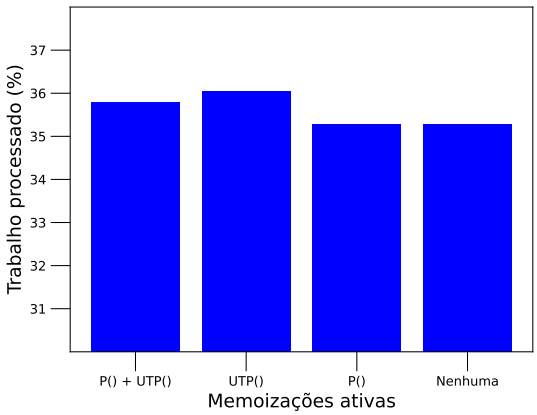


Figura 18. Trabalho processado com memoizações desativadas

de escalonamento para medir o desempenho do algoritmo.

6. Conclusão

Em sincronização híbrida, PLs conservadores e otimistas são combinados em uma mesma simulação computacional. Embora promissora, a abordagem sofre de obstáculos como a ocorrência de eventos mutuamente exclusivos, o que leva ao descarte de mensagens e redução de precisão dos resultados da simulação. [Parizotto and Mello 2022] argumentaram que uma sincronização híbrida poderia ser otimizada com base em duas métricas: número de mensagens e soma de trabalho. Para tanto, foi proposto um algoritmo de escalonamento que seleciona, de forma dinâmica, o subconjunto de mensagens que possui o equilíbrio entre essas duas medidas.

Neste trabalho, foram agregadas duas contribuições à solução proposta em [Parizotto and Mello 2022]. Na primeira delas, ao algoritmo de escalonamento foram adicionadas funcionalidades para memoização das funções $p()$ e $utp()$, visando aproximar os resultados do ponto utópico. A segunda contribuição corresponde à implementação da nova versão da solução na arquitetura DCB, conforme discutido nas perspectivas do próprio artigo-base. A integração do algoritmo proveu o amparo necessário para experimentá-lo em um ambiente real de simulação distribuída, suplementando os testes numéricos apresentados em [Parizotto and Mello 2022].

Os resultados coletados das simulações no DCB evidenciaram que o algoritmo teve desempenho superior à política LTF em simulações com variação no tempo virtual. Além disso, percebeu-se que o algoritmo proposto processou, em média, mais trabalho do que mensagens, e que as simulações distribuídas obtiveram performance similar às contrapartes locais. Experimentos adicionais desativando as funcionalidades adicionadas neste trabalho apontaram um discreto afastamento do ponto utópico, sugerindo que a manutenção dessas modificações beneficia a simulação. De modo geral, a solução proposta foi capaz buscar o equilíbrio entre as duas métricas, e se aproximou mais do ponto utópico do que a diretiva LTF convencional.

6.1. Perspectivas futuras

Conforme comentado, os modelos elaborados para este trabalho pretenderam submeter o algoritmo de escalonamento a situações de estresse artificiais, para aferir a sua resiliência. É desejável que desenvolvimentos futuros tragam modelos mais verossímeis ou pautados em simulações reais, para observar o desempenho do algoritmo nesses cenários.

Trabalhos posteriores podem abordar a impacto das memoizações aqui propostas, complementando ou substituindo os testes supramencionados. A utilização de outras configurações de modelos e a aplicação de métricas distintas, como tempo real de escalonamento ou tempo real de simulação, podem favorecer as análises das contribuições.

Outra possibilidade é afrouxar as medidas de atenuação de aspectos como latência de rede e mensagens em trânsito, para verificar o impacto causado em simulações distribuídas. Implementar outras políticas de escalonamento, afora LTF, e paragonar com a solução proposta também é de interesse em desdobramentos futuros.

Referências

Bryant, R. E. (1977). Simulation of packet communication architecture computer systems. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE.

Carson, J. S. (2004). Introduction to modeling and simulation. In *Proceedings of the 2004 Winter Simulation Conference*. IEEE.

Chandy, K. M. and Misra, J. (1979). Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on software engineering*, SE-5(5):440–452.

Chwif, L. and Medina, A. C. (2006). *Modelagem e simulação de eventos discretos: teoria & aplicações*. Autores, São Paulo, 2. edition.

Eker, A., Arafa, Y., Badawy, A.-H. A., Santhi, N., Eidenbenz, S., and Ponomarev, D. (2021). Load-aware dynamic time synchronization in parallel discrete event simulation. In *Proceedings of the 2021 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 95–105. ACM.

Fujimoto, R. (2015). Parallel and distributed simulation. In *Proceedings of the 2015 Winter Simulation Conference*, pages 45–59. IEEE.

Fujimoto, R. M. (2000). *Parallel and Distribution Simulation Systems*. John Wiley & Sons, Inc., USA.

Fujimoto, R. M. (2003). Distributed simulation systems. In *Proceedings of the 2015 Winter Simulation Conference*, pages 124–134. IEEE.

Jefferson, D. R. (1985). Virtual time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(3):404–425.

Jefferson, D. R. and Barnes, P. D. (2017). Virtual time iii: Unification of conservative and optimistic synchronization in parallel discrete event simulation. In *Proceedings of the 2017 Winter Simulation Conference*, pages 786–797. IEEE.

Kleinberg, J. and Tardos, É. (2006). *Algorithm design*. Pearson Education India.

Lindén, J. (2018). *Synchronization Techniques in Parallel Discrete Event Simulation*. PhD thesis, Acta Universitatis Upsaliensis.

Liu, J., Nicol, D. M., and Tan, K. (2001). Lock-free scheduling of logical processes in parallel simulation. In *Proceedings 15th Workshop on Parallel and Distributed Simulation*, pages 22–31. IEEE.

Mello, B. A. d. and Wagner, F. R. (2002). A standardized co-simulation backbone. In *SoC Design Methodologies*, pages 181–192. Springer.

Parizotto, R. and Mello, B. (2022). Multiobjective scheduling of hybrid synchronization messages. In *Anais do XLIX Seminário Integrado de Software e Hardware*, pages 49–57, Porto Alegre, RS, Brasil. SBC.

Pidd, M. (1994). An introduction to computer simulation. In *Proceedings of the 1994 Winter Simulation Conference*, pages 7–14. IEEE.

Santoro, T. and Quaglia, F. (2010). A low-overhead constant-time ltf scheduler for optimistic simulation systems. In *The IEEE symposium on Computers and Communications*, pages 948–953. IEEE.

Som, T. K. and Sargent, R. G. (1998). A probabilistic event scheduling policy for optimistic parallel discrete event simulation. In *Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS'98 (Cat. No. 98TB100233)*, pages 56–63. IEEE.

Xiao, Z., Unger, B., Simmonds, R., and Cleary, J. (1999). Scheduling critical channels in conservative parallel discrete event simulation. In *Proceedings Thirteenth Workshop on Parallel and Distributed Simulation. PADS 99.(Cat. No. PR00155)*, pages 20–28. IEEE.