CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
Campus Leopoldina - Programa de Pós-Graduação em Automação e Sistemas

Alice Costa de Oliveira

Desenvolvimento de um Simulador de Linhas de Envase de Cerveja para Validação de Regras de Controle de Produção

Leopoldina

2024

Alice Costa de Oliveira

Desenvolvimento de um Simulador de Linhas de Envase de Cerveja para Validação de Regras de Controle de Produção

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Automação e Sistemas do Centro Federal de Educação Tecnológica de Minas Gerais, Campus Leopoldina, como parte dos requisitos para obtenção do título de Mestre em Automação e Sistemas.

Orientador: Prof. Dr. José Geraldo Ribeiro Júnior

Coorientador Prof. Dr. Gustavo Campos Menezes

Leopoldina

2024

Oliveira, Alice Costa de.
O48d Desenvolvimento de um simulador de linhas de envase de cerveja para validação de regras de controle de produção. / Alice Costa de Oliveira. – 2025.
87 f. : il.

Dissertação apresentada ao Programa de Pós-Graduação em Automação e Sistemas (PPGAS), 2025.
Orientador: José Geraldo Ribeiro Júnior.
Coorientador: Gustavo Campos Menezes.
Bibliografia.
Dissertação (mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais. *Campus* Leopoldina.

1. Simulação (Computadores). 2. Sistemas de tempo discreto. 3. Cerveja. 4. Controle da produção. I. Ribeiro Júnior, José Geraldo. II. Menezes, Gustavo Campos. III. Título.

CDU: 004.414.23

Elaboração da ficha catalográfica pela bibliotecária Luzia Adriana Damasceno, CRB 6º 2305 / Cefet/MG c*ampus* Leopoldina.

MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM AUTOMAÇÃO E SISTEMAS - LP

ATA DE DEFESA DE DISSERTAÇÃO Nº 12 / 2024 - PPGAS (11.52.20)

Nº do Protocolo: 23062.063586/2024-70

Leopoldina-MG, 18 de dezembro de 2024.

DISSERTAÇÃO DE MESTRADO

DESENVOLVIMENTO DE UM SIMULADOR DE LINHAS DE ENVASE DE CERVEJA PARA VALIDAÇÃO DE NOVAS REGRAS DE CONTROLE DE PRODUÇÃO

AUTORA: ALICE COSTA DE OLIVEIRA

ORIENTADOR: Professor Doutor José Geraldo Ribeiro Júnior

A Banca Examinadora composta pelos membros abaixo aprovou em 18 de dezembro de 2024 essa Dissertação:

Professor Doutor José Geraldo Ribeiro Júnior (ORIENTADOR)

Centro Federal de Educação Tecnológica de Minas Gerais – CEFET/MG

Professor Doutor Gustavo Campos Menezes (COORIENTADOR)

Centro Federal de Educação Tecnológica de Minas Gerais – CEFET/MG

Professor Doutor Fabiano Pereira Bhering (MEMBRO INTERNO)

Centro Federal de Educação Tecnológica de Minas Gerais – CEFET/MG

Professor Doutor Joventino de Oliveira Campos (MEMBRO EXTERNO)

(Assinado digitalmente em 18/12/2024 17:29)
FABIANO PEREIRA BHERING
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOMLP (11.61.11)
Matrícula: 2556430

(Assinado digitalmente em 18/12/2024 13:11)
GUSTAVO CAMPOS MENEZES
PROFESSOR ENS BASICO TECN TECNOLOGICO
DCCN (11.58)
Matrícula: 1550421

(Assinado digitalmente em 18/12/2024 11:47)
JOSE GERALDO RIBEIRO JUNIOR
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOMLP (11.61.11)
Matrícula: 1322715

(Assinado digitalmente em 18/12/2024 12:51)
JOVENTINO DE OLIVEIRA CAMPOS
ASSINANTE EXTERNO
CPF: ###.###.236-##

Visualize o documento original em **https://sig.cefetmg.br/public/documentos/index.jsp** informando seu número: **12**, ano: **2024**, tipo: **ATA DE DEFESA DE DISSERTAÇÃO**, data de emissão: **18/12/2024** e o código de verificação: **8768b0eed5**

Resumo

Com o advento da Indústria 4.0 e o mercado cada vez mais competitivo, o uso da tecnologia para impulsionar os resultados é uma tendência cada vez mais presente nas empresas. Esse parece ser um caminho sem volta. Junto com este movimento, inúmeras ferramentas têm surgido para facilitar o processo de digitalização dos negócios. A transformação digital tem como propósito melhorar a eficiência dos processos, trazendo a clareza sobre o mesmo e auxiliando na tomada de decisão. Esse movimento impacta diversos setores, inclusive as indústrias. Quando se trata de linhas de produção, os simuladores são uma excelente ferramenta, pois eles têm a capacidade de reproduzir o comportamento de uma linha real, o que permite uma série de análises e testes de cenários, mas sem colocar em risco a produção. Esta dissertação apresenta as etapas da validação de um simulador para linhas de envase de cerveja. Num primeiro momento, o objetivo do simulador é auxiliar na validação de novas estratégias de controle que serão aplicadas nas linhas, mas, posteriormente, ele pode ser utilizado para outras funções, como por exemplo, para verificar a performance se uma nova máquina for adicionada ou verificar qual a configuração mais adequada para a linha em questão. As linhas de envase não são iguais, as máquinas que compõem a linha variam dependendo do tipo de embalagem da cerveja. O simulador deve ser capaz de abstrair essas diferenças e simular qualquer tipo de linha de envase. Nessa dissertação foram realizados os testes de validação da lógica do simulador, que foram importantes para identificação e correção dos erros encontrados.

Palavras-chaves: simulador; simulação a eventos discretos; linha de envase.

Abstract

With the advent of Industry 4.0 and an increasingly competitive market, the use of technology to boost results is a trend that is increasingly present in companies. And this seems to be a path of no return. Along with this movement, numerous tools have emerged to facilitate the process of digitalizing businesses. The purpose of digital transformation is to improve the efficiency of processes, bringing clarity to them and assisting in decision-making. This movement impacts several sectors, including industries. When it comes to production lines, simulators are an excellent tool, as they have the ability to reproduce the behavior of a real line, which allows for a series of analyses and scenario tests, but without putting production at risk. This dissertation presents the validation steps of a simulator for beer packaging lines. Initially, the simulator's objective is to assist in the validation of new control strategies that will be applied to the lines, but later, it can be used for other functions, such as to check performance if a new machine is added or to verify the most appropriate configuration for the line in question. Packaging lines are not the same, the machines that make up the line vary depending on the type of beer packaging. The simulator must be able to abstract these differences and simulate any type of packaing line. This dissertation performed validation tests on the simulator's logic, which were important for identifying and correcting the errors found.

Keywords: simulator; discrete event simulation; packaging line.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CLP	Controlador Lógico Programável
CPH	Contêineres por Hora
CPM	Contêineres por Minuto
ECI	*Empty Container Inspector*
IIoT	*Industrial Internet of Things*
IoT	*Internet of Things*
MTBF	*Mean Time Between Failure*
MTTR	*Mean Time to Repair*
OPC UA	*Open Platform Communications - Unified Architecture*
POO	Programação Orientada a Objetos
SALBP	*Simple Assembly Line Balancing Problem*
SCADA	*Supervisory Control And Data Acquisition*
SED	Simulação a Eventos Discretos
UTC	Tempo Universal Coordenado [1]

[1] UTC foi a sigla escolhida para ser utilizada como padrão com o intuito de evitar confusão entre diferentes idiomas (National Institute of Standards and Technology - NIST, 2024)

Sumário

1 Introdução

A Indústria 4.0 é marcada pela digitalização nas indústrias, com o intuito de melhorar a produção e a gestão, tornando as linhas mais eficientes e munindo os gestores com ferramentas que ajudam na tomada de decisão (Calış Duman; AKDEMIR, 2021). Dentro desse contexto surgiu uma vertente chamada de Internet das Coisas Industrial, do inglês, *Industrial Internet of Things* (IIoT). A principal diferença entre a IoT (*Internet of Things*) e a IIoT é a área de aplicação; enquanto a primeira é focada nos consumidores finais e apresenta pouco risco, a segunda tem foco nas indústrias e qualquer falha pode causar perdas significativas. A IIoT conecta os equipamentos das fábricas, faz a aquisição de um grande volume de dados e tem o foco na confiabilidade. Dentre os objetivos estão a melhora da eficiência e da produtividade e a possibilidade do monitoramento e tomada de decisão em tempo real (BABAYIGIT; ABUBAKER, 2024).

Indústrias de diferentes setores conseguem se beneficiar com as novas tecnologias. Uma ferramenta que tem ganhado espaço são os simuladores das linhas de produção. Os simuladores permitem que a linha seja modelada de forma que reproduza o comportamento real no ambiente virtual. No mercado existem diversas ferramentas para facilitar essa modelagem. Com o modelo, é possível realizar inúmeras análises, como verificar a performance da linha e testar cenários para identificar aquele que irá trazer mais resultados. Uma grande vantagem é o fato das simulações não oferecerem riscos à planta e não terem grandes custos associados, como desperdício de insumos e paradas na produção. Na literatura encontra-se vários exemplos da aplicação da simulação a eventos discretos (SED) para simular linhas de produção, alguns deles serão apresentados na Seção 2.2.

As simulações também podem ser utilizadas no comissionamento de novos sistemas de controle. Uma vez que se tem a modelagem da linha, é possível utilizá-la para testar e validar novas lógicas de controle de uma forma mais rápida e segura (LAURINDO et al., 2017).

O presente trabalho tem como plano de fundo a indústria cervejeira e, mais precisamente, das linhas de envase de cerveja, também chamadas de *packaging line*. Como o próprio nome já diz, essa é a etapa em que a cerveja é envasada e fica pronta para a comercialização. As linhas de envase são linhas de produção contínua, pouco flexíveis e têm longos tempos de *setup*, quando há a troca de produto, e dificuldade de produzir pouca quantidade. O principal gargalo das linhas de envase são as enchedoras, *filler* em inglês, que é a máquina que deve balizar a programação de toda a linha (NICOLETTI et al., 2016).

Uma linha de envase é composta por um conjunto de máquinas que têm papéis

bem definidos, e tem como objetivo fracionar o produto nas embalagens e deixá-lo pronto para a comercialização. As linhas de envase podem ser usadas para diversos produtos, como por exemplo: alimentos, cosméticos, produtos farmacêuticos e bebidas. Este trabalho tem como foco linhas de envase de cerveja.

As máquinas das linhas de envase são ligadas por esteiras e/ou mesas de acumulação, que serão chamados de *buffers* neste trabalho. Os *buffers* são responsáveis por transportar as unidades entre as máquinas. Esses equipamentos servem como armazenamento temporário e são fundamentais na dinâmica das linhas. Quando uma máquina apresenta falha e para, tem-se o tempo para encher completamente o *buffer* até que a máquina anterior seja impactada e pare de produzir, e assim por diante, até que toda a linha pare devido àquela falha. Dessa forma, quando uma nova linha é projetada, o tamanho dos *buffers* é parte importante do projeto. A Figura 1.1 exemplifica um trecho de uma linha de envase.

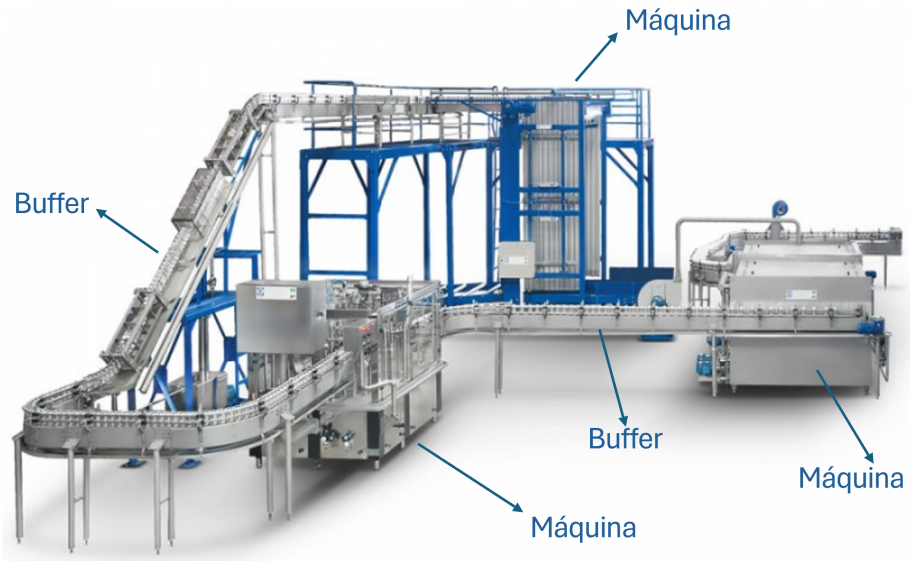

Figura 1.1 – Trecho de uma linha de envase. Adaptado de: Zegla (2024).

Existem três tipos de linha de envase de cerveja, uma para latas, uma para garrafas retornáveis e uma para as não retornáveis. Para contemplar os diferentes tamanhos, a configuração das máquinas é alterada de acordo com a necessidade. As unidades que circulam na linha, que podem ser garrafas ou latas, são chamadas de contêineres. A velocidade das máquinas comumente é medida em contêineres por hora (CPH), podendo também ser representada em contêineres por minuto (CPM).

A cerveja é envasada em latas e garrafas de vidro e, em ambos os casos, existe mais de um tamanho disponível. As latas podem ser de 269 ml, 350 ml, 473 ml ou 550ml. Já as garrafas são divididas em dois grupos, as retornáveis, que são as com capacidade de 300 ml, 600 ml e 1 L, e as não retornáveis, conhecidas como *long neck*, que têm capacidade

de 330 ml e 355 ml. A Figura 1.2 exemplifica sete dos tamanhos citados.

Figura 1.2 – Diferentes tamanhos de cervejas. Adaptado de: Observatório da Embalagem (2018).

1.1 Definição do Problema

Uma cervejaria, que por questões de confidencialidade não terá seu nome revelado, investe no desenvolvimento de produtos internos que auxiliem no monitoramento e otimização dos resultados de suas linhas de produção. No caso de ferramentas de monitoramento, em que o software não vai atuar no sistema real e vai apenas obter os dados, os testes de validação podem ser feitos em produção sem oferecer riscos à planta e à sua performance. Já nas ferramentas que atuam no sistema, é de extrema importância que o software seja previamente testado, para que quando for instalado em produção apresente o funcionamento esperado, sem comprometer os resultados daquela linha. Atualmente, a cervejaria não utiliza simuladores para testar esse tipo de aplicação, e é nesse contexto que surgiu o tema dessa dissertação.

Um dos produtos internos da empresa é o PaCo[1], de *Packaging Control*, que é um sistema de controle de linhas de envase e será devidamente apresentado no Capítulo 2. O PaCo é uma ferramenta para auxiliar no balanceamento das linhas e tem como objetivo melhorar os indicadores da mesma. Esse é um produto que monitora e atua diretamente na linha, alterando a velocidade das enchedoras. Essa ferramenta já foi instalada em diversas plantas e seus resultados já foram comprovados, mas o software continua em atualização. Como cada planta tem suas especificidades, os usuários da ferramenta destacam pontos de melhorias para o que já está implementado e reportam novas funcionalidades que gostariam que fossem adicionadas. Atualmente, toda nova versão do software que é disponibilizada, só é propriamente testada em produção, o que é um risco para a empresa. Quando algum erro é detectado, a equipe de desenvolvimento precisa agir prontamente

[1]Nome fictício

para corrigi-lo e, em muitos casos, não há como reproduzi-lo de maneira manual. Isso acontece porque a empresa não detém um simulador capaz de emular uma linha de envase real, que permita um teste mais robusto e que se assemelhe ao ambiente de produção. O objetivo deste trabalho é desenvolver um simulador que emule a lógica básica de uma linha de envase para auxiliar, em um segundo momento, nos testes do software do PaCo. Posteriormente, a expectativa é que esse simulador possa ser utilizado no contexto de outros produtos da empresa.

1.2 Objetivos

A seguir estão descritos o objetivo geral e os objetivos específicos deste trabalho.

1.2.1 Objetivo Geral

O principal objetivo deste trabalho é desenvolver uma ferramenta para simulação de linhas de envase de cerveja, que seja capaz de reproduzir o comportamento de uma linha real, no que diz respeito à dinâmica do comportamento das máquinas e dos *buffers*. O intuito é que essa ferramenta auxilie o time de desenvolvimento dos produtos internos da empresa na identificação e correção de erros antes de que sejam aplicados nas linhas de produção.

1.2.2 Objetivos Específicos

Dentre os objetivos específicos estão:

- Reunir as principais ferramentas de simulação disponíveis no mercado e na literatura e verificar como elas podem ser utilizadas nesta dissertação, com o intuito de buscar referências para o desenvolvimento do simulador;
- Analisar o material fornecido pela cervejaria sobre linhas de envase, para identificar quais comportamentos devem estar representados no simulador;
- Definir as ferramentas para desenvolvimento do simulador;
- Desenvolver um simulador *open-source* que represente uma linha de envase genérica e que seja capaz de reproduzir o comportamento real da linha, principalmente no que diz respeito o funcionamento das máquinas e *buffers*;
- Definir uma característica estocástica para a linha, a fim que a simulação se assemelhe ao real;
- Validar a lógica implementada no simulador.

1.3 Principais Contribuições

A principal contribuição desta dissertação está no desenvolvimento de um simulador capaz de reproduzir o comportamento básico de uma linha de envase de cerveja, já que não é possível ter uma linha real disponível para isso. Além da disponibilização do simulador, é possível citar as seguintes contribuições:

- Possibilidade de configurar a linha de envase de acordo com o cenário proposto. É possível habilitar e desabilitar máquinas de acordo com a linha que se deseja simular;
- É possível configurar alguns parâmetros das máquinas, como a velocidade nominal, e o tamanho dos *buffers*;
- É possível testar cenários com linhas simples e com bifurcações para analisar a dinâmica da linha;
- Os dados de simulação são salvos em um banco de dados de série temporal, que podem ser facilmente exportados para a análise dos dados;
- É possível ter uma simulação determinística ou estocástica.

1.4 Organização

O restante desse trabalho está dividido conforme descrito a seguir:

- O Capítulo 2 apresenta a fundamentação teórica dos conceitos e revisão bibliográfica utilizados no desenvolvimento deste trabalho.
- O Capítulo 3 descreve a metodologia utilizada e desafios encontrados no desenvolvimento do trabalho. Neste capítulo o simulador proposto é apresentado.
- O Capítulo 4 apresenta os testes realizados para validar o funcionamento do simulador proposto.
- Por fim, o Capítulo 5 apresenta a conclusão do trabalho e a sugestão de trabalhos futuros.

2 Revisão Bibliográfica

Este capítulo apresenta os principais conceitos e trabalhos relacionados ao assunto dessa dissertação. A Seção 2.1 apresenta o processo de otimização e balanceamento de linhas de produção e destaca como os simuladores podem ser utilizados para auxiliar nesse processo. A Seção 2.2 apresenta as simulações a eventos discretos, muito utilizadas para modelar linhas de produção, e cita os principais simuladores disponíveis no mercado. A Seção 2.3 apresenta os principais trabalhos relacionados ao temas apresentados neste capítulo. Por fim, a Seção 2.4 apresenta a fundamentação teórica, que descreve as linhas de envase, apresenta o projeto PaCo, mencionado na seção 1.1, e o simulador que foi utilizado como base nesta dissertação.

2.1 Otimização e Balanceamento de Linhas de Produção

As linhas de produção são configuradas e otimizadas quando são projetadas pelo fornecedor, mas, com o passar dos anos, essa programação deixa de ser suficiente e a linha perde eficiência. Isso pode acontecer pela ação do tempo, com o desgaste dos equipamentos, ou por mudanças significativas, como, por exemplo, a substituição ou inserção de uma máquina. Isso faz com que as empresas, sempre pensando na performance e melhores resultados, invistam no desenvolvimento de soluções específicas para sua demanda.

Na literatura existe o conceito de balanceamento de linha de montagem, do inglês, *assembly line optimization*, que consiste em otimizar o funcionamento da linha para maximizar as saídas para os recursos disponíveis (BOYSEN; SCHULZE; SCHOLL, 2022). Ou seja, ajustar a linha para que se tenha o melhor resultado utilizando os equipamentos e recursos que já estão disponíveis. Existe o formalismo que trata uma linha simples, conhecido como *Simple Assembly Line Balancing Problem* (SALBP) e, a partir deste foram criados outros, para contemplar características mais complexas, como por exemplo, paralelismo de tarefas e/ou máquinas e linhas no formato U (Borsato Pinhão; IGNACIO; COELHO, 2022).

O projeto do balanceamento de linhas de produção é um trabalho complexo, que exige bastante conhecimento técnico, tanto dos conceitos quanto do funcionamento da linha, e que pode demorar meses para ser desenvolvido. Uma vez que se tem as regras para a otimização da linha, é necessário implementar o controle e validá-lo. Essa etapa pode ser um período de incertezas, pois, geralmente, as empresas não detêm um simulador para ser utilizado na validação dessas novas ferramentas. A única alternativa é testar na própria linha de produção, onde qualquer falha pode significar alguns milhares de reais

de prejuízo. Dessa forma, os simuladores podem ser uma excelente ferramenta, que vão acelerar o desenvolvimento e validação de novas lógicas de controle, de uma maneira mais rápida e segura.

2.2 Simulação a eventos discretos

Os sistemas a eventos discretos são sistemas discretos que sofrem alterações em seu estado apenas quando há a ocorrência de um um evento (LIN et al., 2020). As simulações a eventos discretos (SED) são estocásticas, dinâmicas e regidas por eventos, e são utilizadas para modelar sistemas reais complexos. As simulações são rápidas, permitem que várias hipóteses sejam testadas sem oferecer risco algum para o sistema e os resultados auxiliam na tomada de decisão. SED são utilizadas em diversos setores desde a década de 1950 e, devido suas características, conseguem modelar muito bem linhas de produção, o que faz com que sejam bastante utilizadas por esse setor (FORBUS; BERLEANT, 2022).

Existem no mercado várias ferramentas de automação que utilizam a simulação a eventos discretos, que variam desde as mais simples até as mais completas, que contêm uma interface, permitem gerar animações e geram os mais diversos resultados. Na maior parte dos casos, essas ferramentas são pagas e as licenças são caras, o que faz com que apenas as grandes empresas tenham acesso.

Na fase inicial do trabalho, encontrar uma ferramenta para auxiliar no desenvolvimento do simulador foi um desafio, já que ela deveria ser *open-source* ou conter uma licença gratuita para estudantes. O artigo publicado por (LANG et al., 2021) apresenta um estudo comparativo entre softwares de simulação a eventos discretos comerciais e os *open-source*. Foram analisados dois softwares comerciais (ARENA e *Plant Simualtion*) e três softwares *open-source* (Salabim, JaamSim e CloudSim). O objetivo do artigo é fornecer informações sobre ferramentas *open-source* que possam ser utilizadas em trabalhos acadêmicos e em empresas que não conseguem arcar com os custos dos software licenciados. O artigo destaca os pontos positivos e negativos de cada um dos softwares.

A seguir estão listados os softwares que foram levados em consideração na pesquisa dessa dissertação:

- Os softwares *Arena Simulation Software* (Rockwell Automation, 2024), *FlexSim* (Autodesk Company, 2024), *Plant Simulation* (SIEMENS, 2024) e o *anylogic* (AnyLogic Company, 2024) são softwares bem semelhantes. Todos possuem uma interface gráfica bem completa, com uma biblioteca com vários objetos pré-definidos para auxiliar no desenvolvimento do modelo e visualização 3D. Os quatro softwares permitem a execução de vários cenários de teste e possuem várias ferramentas de análises para auxiliar na tomada de decisão. Todos os software são pagos para uso

comercial, mas disponibilizam versões gratuitas para estudantes, podendo ser com recursos limitados ou não.

- JaamSim (JaamSim, 2024) é um software *open-source*. Assim como os citados anteriormente, possui uma interface e biblioteca com objetos pré-definidos. Existe também a versão Pro, que é paga e contém algumas ferramentas adicionais, como uma lista de objetos mais extensa.
- Salabim (Salabim, 2024) e ProdSim (ProdSim, 2024) são duas bibliotecas de simulação a eventos discretos em Python. Não possuem interface gráfica, portanto, toda a configuração é feita por código e não possuem uma biblioteca de objetos pronta. O Salabim permite criação de animações em 2D e 3D. Por serem bibliotecas, podem ser integradas à outras aplicações em Python com facilidade.

2.3 Estado da Arte

O artigo de Kliment et al. (2020) tem como objetivo melhorar a eficiência e qualidade dos produtos de uma linha de produção do ramo alimentício. A empresa empacota saladas, bacalhau e outros tipos de peixe. O software utilizado foi o *Tecnomatix Plant Simulation*, que permitiu a análise de diferentes cenários. Dentre os resultados, foram propostas mudanças físicas na linha e mudanças no controle para aumentar a produtividade.

No artigo de Prado, Mendonça e Dutra (2023) foi desenvolvida uma simulação de uma linha de envase de bebidas localizada no estado de Goiás. Neste caso, o software escolhido foi o *Anylogic*, que permitiu uma análise quantitativa dos resultados. O trabalho identificou de alguns gargalos e melhorias físicas a serem implementadas na linha.

Os autores Ujam e Godwin (2018) realizaram um estudo de caso em uma cervejaria na Nigéria. O objetivo do trabalho era reduzir o número de paradas e melhorar a performance da linha, otimizando os recursos já disponíveis. Após algumas semanas de aquisição de dados, uma série de parâmetros foram calculados, como por exemplo, a eficiência da linha, das máquinas e dos *buffers*, para identificar onde estavam os problemas. Em seguida, foi criado um modelo da linha no software *Tecnomatix Plant Simulation*, para auxiliar no entendimento do comportamento do sistema. Todas as etapas do estudo geraram resultados importantes para a otimização das linhas de envase analisadas.

O artigo de Achkar et al. (2020) propõe um modelo de simulação a eventos discretos utilizando o software Simio (Simio, 2024), que permite a emulação e predição do comportamento de sistemas complexos. O intuito do trabalho foi construir essa ferramenta para auxiliar uma cervejaria argentina na tomada de decisão de um nova linha de envase tendo como foco a logística. Foram levadas em consideração diversas variáveis, como pro-

gramação de produção, armazenamento interno, logística reversa das garrafas retornáveis, entre outras.

Os autores de Dereje, Abeya e Gopal (2020) criaram um modelo de simulação, utilizando o software Arena (Rockwell Automation, 2024), para uma linha de envase de água mineral em garrafas de vidro retornáveis de uma empresa na Etiópia. O objetivo era a identificação de gargalos e proposição de soluções para melhorar a performance das linhas, de acordo com os resultados dos cenários de testes implementados.

Os simuladores das linhas de produção podem ser empregados de forma análoga aos trabalhos mencionados, sendo utilizados, principalmente, para a realização de testes de cenários, com o objetivo de aprimorar o desempenho e os resultados da linha. Outra aplicação relevante, que motivou este trabalho, é a utilização dos simuladores como ferramenta de apoio no desenvolvimento de novos sistemas de controle e/ou monitoramento para as linhas de produção.

2.4 Fundamentação Teórica

Esta seção apresenta os principais conceitos utilizados e contextualiza este trabalho. A Seção 2.4.1 define os quatro estados em que a máquina pode estar durante a simulação. Na Seção 2.4.2 são apresentados dois parâmetros de performance, que serão utilizados como entradas da simulação. Na subseção 2.4.3 é apresentado o BrewSim, um simulador antigo da cervejaria que foi utilizado como base para o simulador dessa dissertação. E por fim, a Seção 2.4.4 apresenta o PaCo, projeto interno da cervejaria que controla as linhas de envase de cerveja.

2.4.1 Estados das Máquinas

As máquinas das linhas de envase, de maneira simplificada, têm quatro estados: (1) operando, (2) falta de contêineres, (3) bloqueada e (4) parada. O significado de cada um deles está descrito a seguir:

- Operando (*running*): estado em que a máquina está operando normalmente.
- Falta de contêineres (*starved*): estado em que a máquina não apresenta nenhuma falha interna, mas o *buffer* de entrada, aquele localizado logo antes da máquina, está vazio. A máquina para por falta de contêineres disponíveis.
- Bloqueada (*blocked*): estado em que a máquina não apresenta nenhuma falha interna, mas o *buffer* de saída, aquele localizado logo após a máquina, está completamente cheio. Neste caso, a máquina para por falta de espaço disponível para os contêineres.

- Parada (*stopped*): estado em que a máquina está parada devido à alguma falha interna ou a linha toda está parada.

2.4.2 MTBF e MTTR

Na literatura existem alguns parâmetros usados para medir a performance das máquinas e das linhas de produção. Esses parâmetros podem ser utilizados para diferentes propósitos, como por exemplo: medir a performance de uma linha após o processo de modernização (MICHLOWICZ, 2022), medir a performance de máquinas críticas (DHIMAN; KUMAR, 2023), desenvolver sistemas de manutenção preventiva para linhas de produção (C.E. O. O. OBIUKWU, 2023), entre outros.

Dentre os parâmetros existentes, este trabalho destaca dois: o MTBF (do inglês, *mean time between failures*) que mede o tempo médio entre falhas e o MTTR (do inglês, *mean time to repair*) que é tempo médio de reparo. As Equações 2.1 e 2.2 apresentam os cálculos para obtenção de cada um deles (ZENG; SHAO; HAO, 2021).

$$MTBF = \frac{\text{tempo total da máquina em operação}}{\text{número de falhas}} \tag{2.1}$$

$$MTTR = \frac{\text{tempo total da máquina em falha}}{\text{número de falhas}} \tag{2.2}$$

Neste trabalho, MTBF e MTTR são parâmetros de entradas do simulador e definem o momento de início e a duração de paradas aleatórias das máquinas. Como são parâmetros de performance, eles são calculados com os dados históricos de paradas de máquina, porém, no simulador é utilizada a lógica inversa: a partir dos valores médios informados, as paradas por falha serão induzidas no simulador. O valor do MTBF irá definir o início das falhas e o MTTR irá definir o fim delas. A lógica implementada no simulador será apresentada posteriormente na Seção 3.4.

2.4.3 Simulador Base

No contexto de um outro projeto interno da cervejaria, uma empresa terceira foi contratada para desenvolver um simulador capaz de reproduzir o comportamento de suas linhas de produção de cerveja e envase. O simulador seria utilizado para testar e validar o projeto. Neste trabalho, esse simulador será chamado de BrewSim[1], *Brewery Simulator*.

O BrewSim foi desenvolvido em um software de automação industrial chamado Ignition (Inductive Automation, 2024). O Ignition consiste em um sistema SCADA (*Su-*

[1]Nome fictício

pervisory Control And Data Acquisition) que permite o projeto e a aquisição de dados de qualquer aplicação industrial, impulsionando assim a transformação digital nas indústrias. O Ignition é um software proprietário na qual é necessário a aquisição de uma licença para o uso. Porém, é possível fazer o download do software completo no site de forma gratuita. A restrição da versão gratuita é que a licença fica ativa apenas por duas horas, após esse tempo ela deve ser renovada e o teste reiniciado. Não há um limite para o número de renovações da licença.

Além do Ignition, é necessário instalar o SQL Server (Microsoft, 2024) e o Python (Python, 2024) para utilizar o simulador BrewSim. O primeiro é o banco de dados da aplicação e o segundo serve para executar uma API que faz alguns cálculos mais complexos. É importante ressaltar que o SQL Server também é um software proprietário e necessita de licença.

O Ignition é um software completo, que permite a adição de lógica e comportamentos em diversos lugares, como por exemplo o comportamento do *click* de um botão e o valor de uma *tag* de dados, que é típico de um software SCADA. Com isso, não é possível ter uma visão geral de tudo que está implementado, pois não existe uma lista de arquivos com todos os códigos. Em alguns casos, como o exemplo do botão e da *tag* de dado, como o código define o comportamento do objeto, para acessá-lo é necessário abrir a janela de propriedades objeto. Somado a isso, a falta de documentação do BrewSim dificultou o processo de validação do funcionamento do simulador. Não haviam documentos que apresentassem o simulador e suas funcionalidades, nem manuais de configuração e utilização do mesmo. Também não haviam documentos que descrevessem como foi feita a modelagem da simulação.

Como dito anteriormente, o simulador é capaz de simular dois processos: a fabricação dos lotes (fabricação da cerveja) e as linhas de envase. Foram criadas as telas de configuração e monitoramento para ambos os casos. Na tela de configuração das linhas de envase, é possível criar uma linha de acordo com a necessidade, permitindo adicionar mais de uma máquina por grupo. A tela de monitoramento tem algumas opções para configurar a simulação e um botão para iniciá-la. Uma vez iniciada, uma representação da linha é criada na tela onde é possível acompanhar o estado das máquinas, que pode variar entre os estados descritos na subseção 2.4.1 e a porcentagem de acumulação dos *buffers*. O simulador permite que a simulação seja executada de maneira (1) manual, em que o usuário pode alterar o estado das máquinas como julgar necessário, e (2) automática, em que o simulador é responsável por comandar o estado das máquinas.

Após a utilização no projeto em que foi concebido, o simulador não foi aproveitado por outras equipes dentro da companhia. Alguns fatores levaram à pouca utilização do simulador BrewSim. É possível citar por exemplo: licença proprietária, dificuldade de instalação e de compartilhamento do software entre os projetos da empresa, falta de

documentação e falta de pessoas capacitadas para dar manutenção.

Como o objetivo deste trabalho é a criação de um simulador capaz de auxiliar e facilitar o desenvolvimento de outras aplicações dentro da empresa, inicialmente foi considerada a ideia de dar continuidade com BrewSim. Dessa forma, a lógica do simulador seria validada e atualizada de acordo com a necessidade. Porém, levando em consideração a utilização de softwares proprietários e a falta de documentação do simulador, que dificultou o entendimento e controle total da ferramenta, essa ideia foi descartada. Com o intuito de aproveitar a lógica implementada no BrewSim, levando em consideração que houve investimento da empresa para o desenvolvimento do mesmo, ele foi utilizado como base no simulador proposto nesta dissertação.

2.4.4 Packaging Control - PaCo

Conforme apresentado na Seção 1.1, a cervejaria desenvolve um produto interno para balancear e otimizar as linhas de envase. O PaCo é utilizado em diversas plantas da cervejaria, já apresenta resultados positivos comprovados e está em expansão, tanto na utilização, sendo instalado em novas plantas, quanto no desenvolvimento de novas funcionalidades. Foi projetado para ser utilizado *on premise*, ou seja, é instalado em uma máquina virtual disponível na infraestrutura local da cervejaria. O PaCo deve ter acesso à rede interna da cervejaria, a fim de que possa ler e escrever os dados necessários para seu funcionamento. A aplicação utiliza o *Docker* (Docker, 2024) e contém cinco *containers* principais, conforme descrito a seguir.

- PostegreSQL (PostgreSQL, 2024): banco de dados para salvar os dados de configuração da linha;
- InfluxDB (InfluxData, 2024): banco de dados de série temporal para salvar os dados de execução da aplicação;
- API: responsável por buscar os dados nos bancos de dados e responder a interface;
- Interface: página *web* onde é possível configurar e monitorar a aplicação;
- Controle: executa a lógica de controle de acordo com as configurações salvas no banco de dados.

Para a aquisição dos dados em tempo real da linha, o PaCo utiliza o protocolo de comunicação de automação industrial chamado OPC UA, do inglês *Open Platform Communications – Unified Architecture* (OPC Foundation, 2024). Os CLPs atualizam em tempo real o servidor OPC UA com as informações da linha, esses valores são salvos em *tags* de dados. O PaCo, por sua vez, acessa o servidor OPC UA para ler ou escrever essas *tags* de dados.

Na interface, o usuário cria e configura as linhas de envase a serem controladas.

É necessário definir quais máquinas fazem parte da linha e informar os endereços das *tags* que contêm os dados necessários, como por exemplo, o estado de cada máquina.

A aplicação contém sete blocos de controle pré-definidos, onde cada um deles contém as condições que ativam e desativam o bloco. Essas condições podem monitorar o estado de determinadas máquinas ou a porcentagem de acumulação de algum *buffer*. O usuário deve configurar todas as condições, de ativação e desativação, de acordo com as características da planta. É importante que esses parâmetros estejam bem ajustados para se obter melhores resultados.

O objetivo principal do PaCo é evitar microparadas da máquina gargalo, que geralmente é a enchedora. São consideradas microparadas, as paradas de máquina que tem duração menor que cinco minutos. O PaCo contém blocos de controle que definem cenários em que se deseja alterar a velocidade dessa máquina, que normalmente opera na velocidade nominal, com o intuito de evitar paradas causadas por faltas de contêineres no *buffer* de entrada (estado *starved*) ou falta de espaço no *buffer* de saída (estado *blocked*). Durante a execução, todos os blocos de controle são analisados e, de acordo com o bloco ativo, é definido um *setpoint* de velocidade para a enchedora. Esse *setpoint* é usado para colocar um valor menor ou maior que a velocidade nominal, fazendo a máquina operar em subvelocidade ou sobrevelocidade, respectivamente.

A Figura 2.1 apresenta, de forma simplificada, o fluxo do controle do PaCo.

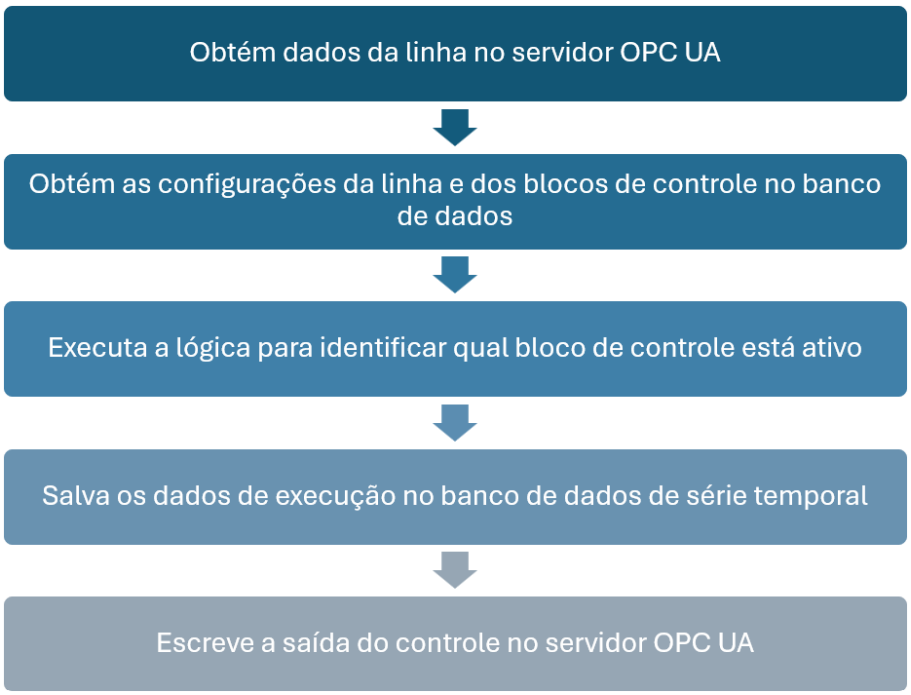

Figura 2.1 – Fluxo do PaCo.

O PaCo é um software em desenvolvimento e, de tempos em tempos, uma nova

versão é lançada. As novas funcionalidades são testadas em ambiente de desenvolvimento de forma limitada, já que a equipe de desenvolvimento não dispõe de uma ferramenta de simulação de linhas de envase e os testes têm que ser feitos manualmente. Dessa forma, a validação final só é feita quando a nova versão é aplicada em produção. Esse processo não é muito eficaz e coloca a produção da linha em risco, pois, em alguns casos o erro inviabiliza a utilização do PaCo na linha. Além disso, uma vez que o problema é detectado em produção, é mais trabalhoso encontrar a causa e, em várias situações, não é possível reproduzir a falha em ambiente de desenvolvimento.

3 Desenvolvimento do Simulador Proposto

O simulador proposto nesta dissertação recebeu o nome de PLSim, de *Packaging Line Simulator*. Para o desenvolvimento do PLSim, a primeira etapa foi uma longa investigação para entender o funcionamento tanto do Ignition quando do simulador BrewSim. Para isso, foi preciso reunir a documentação disponível, que consistia em apenas um manual de instalação, para operacionalizar o simulador em um computador. Após a instalação dos softwares, foi o momento de entender como acessar os códigos do BrewSim e entender como o Ignition orquestra tudo. Essa etapa foi importante para identificar quais as funcionalidades disponíveis no simulador, analisar se a lógica implementada era satisfatória e entender como ele poderia ser utilizado para auxiliar a equipe de desenvolvimento nos testes do software do PaCo.

Apesar do simulador BrewSim conter uma interface visual para acompanhamento da simulação, apenas a observação da evolução de um teste não é suficiente para julgar se a simulação está adequada ou não, é necessário entender a regras implementadas. Para isso, foi preciso identificar no código a função, ou conjunto de funções, que contém a lógica principal que define o comportamento das máquinas e *buffers*. O comportamento das máquinas está descrito em uma função extensa, chamada *packagingFSM*, que chama outras funções auxiliares e que leva em consideração variáveis que são salvas em tags do servidor OPC UA.

Com o objetivo de entender os detalhes da lógica implementada no simulador BrewSim, a sua função principal, bem como as funções auxiliares, foram traduzidas em fluxogramas que facilitaram o entendimento do código e da lógica proposta. A primeira versão dos fluxogramas foi uma representação fiel do código, onde todas as variáveis e condições foram incluídas. Devido ao seu nível de detalhamento, a primeira versão ficou muito extensa e de difícil entendimento, com isso, foi feita uma versão simplificada, com um nível de abstração mais alto, para apresentação neste trabalho. A Figura 3.1 apresenta a versão simplificada do fluxograma que representa a lógica do simulador BrewSim.

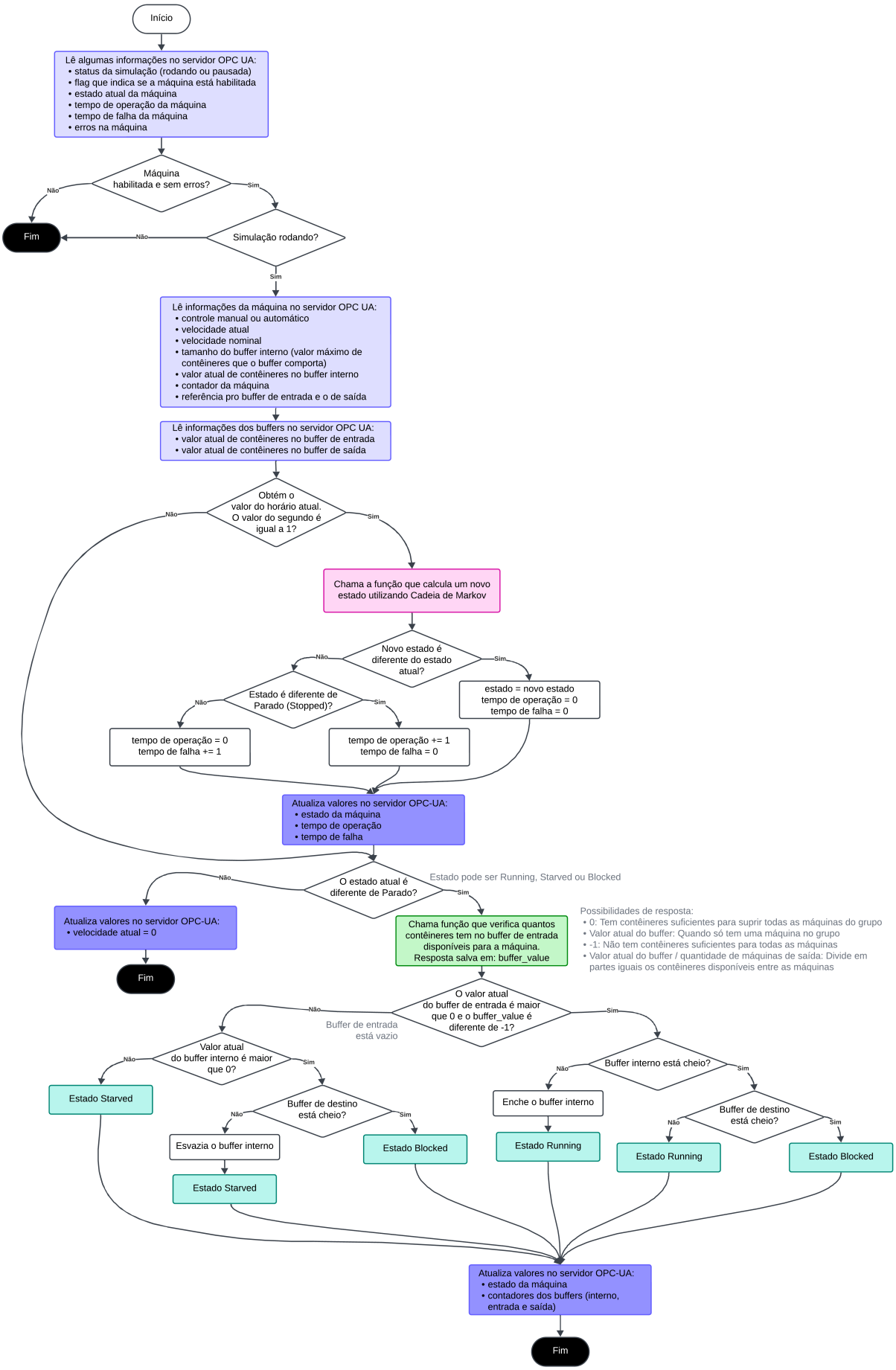

Figura 3.1 – Fluxograma simplificado que apresenta a lógica implementada no simulador BrewSim.

O fluxo apresentado na Figura 3.1 é a representação da lógica de um método implementado no BrewSim, que é executado para cada uma das máquinas a cada um segundo. A lógica começa lendo no servidor OPC UA a flag que indica se a simulação está rodando ou em pausa e algumas informações da máquina, verifica se a mesma está habilitada, lê o estado atual, entre outros. Se a máquina não está habilitada ou a simulação está em pausa, o método é finalizado, caso contrário, a lógica continua. Em seguida, o simulador lê no servidor OPC UA outras informações da máquina e informações do *buffer*, conforme listado na Figura. A cada minuto é chamada uma função que determina um novo estado para a máquina utilizando Cadeia de Markov. Se o novo estado é diferente do anterior, os tempos de operação são resetados e o estado atual atualizado. Se os estados forem iguais, o contador de tempo de operação ou de falha é atualizado. Na sequência o código verifica se o estado atual é Parado, se sim, a velocidade é atualizada para zero e o método encerrado, se não, a lógica continua. O código chama uma outra função que vai calcular quantos contêineres do *buffer* de entrada estão disponíveis para a máquina, de acordo com a saída o código decide se a máquina vai para o estado *Starved*, *Blocked* ou *Running*. Por fim, o estado da máquina e os contadores dos *buffers* são atualizados.

Após o período de entendimento da lógica do BrewSim, foi preciso decidir entre dar continuidade a ele ou começar um novo em um ambiente de desenvolvimento mais controlado, tendo o BrewSim como base. Apesar das vantagens de se utilizar o BrewSim, que foi implementado no Ignition e que, por ser um software de automação, já contém uma série de funcionalidades e integrações que facilitam o desenvolvimento de um simulador como esse, optou-se por desenvolver um simulador do zero. O motivador principal para essa decisão foi o fato de que o BrewSim tem muitas outras funcionalidades além do simulador de linhas de envase, tornando-o muito mais complexo para a necessidade. Outro fator que influenciou na decisão foi a falta de documentação do BrewSim, que impossibilitou o entendimento do funcionamento de partes importantes do simulador, como por exemplo, em como fazer a configuração de uma linha, quais dados são salvos nos banco de dados, como obter os resultados da simulação para além da interface visual e o que significa cada uma das *tags* disponíveis no servidor OPC UA.

Como o PLSim foi uma demanda que nasceu dentro do PaCo, nesse primeiro momento ele foi desenvolvido com base nas linhas de envase e nas tecnologias que o PaCo utiliza. Algumas das principais semelhanças entre o PLSim e o PaCo estão listadas a seguir:

- Ambos foram desenvolvidos em Python;
- Contêm a mesma estrutura para as linhas de envase (máximo de 31 máquinas - mais detalhes a seguir);
- Utilizam o InfluxDB, banco de dados de série temporal, para salvar os dados de simulação.

As funções do BrewSim que foram identificadas por executar a lógica da simulação das linhas de envase foram reescritas no repositório do novo simulador em Python e todo o comportamento original foi mantido. Durante a fase de entendimento e desenho dos fluxogramas das funções algumas melhorias foram identificadas, como por exemplo, adicionar uma documentação adequada ao código, excesso de código repetido que poderia ser evitado, haviam linhas de código comentadas, condições sem sentido, entre outros. Nessa fase, com o intuito de manter a correlação e rastreabilidade entre os códigos e o fluxograma, o código não foi otimizado e a estrutura principal do código foi mantida.

O PLSim foi desenvolvido utilizando programação orientada a objetos (POO) e contém três classes:

- *Machine*: contém o comportamento das máquinas, que é o que rege a simulação;
- *AccumulationBuffer*: contém o comportamento dos *buffers*;
- *Line*: responsável por criar todos os objetos de máquinas e *buffers*, bem como a relação entre eles, e em rodar a simulação de acordo com as variáveis de entrada.

3.1 Configuração da Simulação

O PLSim contém duas possibilidades de execução. A primeira tem como objetivo imitar o tempo de funcionamento de uma linha real, que é um processo contínuo, em que é possível acompanhar as mudanças do sistema a medida que acontecem. Esse modelo consiste em um *loop* infinito que executa as funções a cada 1 segundo. A segunda é regida pelo número de ciclos, que tem início e fim de acordo com o número de ciclos informado. Nesse caso, não existe o *delay* de 1 segundo e horas de simulação são executadas em poucos segundos. Em ambos os casos é necessário informar o número de unidades de entrada, que é a quantidade de contêineres que a linha irá processar. A velocidades das máquinas são definidas em taxas que definem quantos contêineres por segundo cada uma delas é capaz de processar.

Independente do modo de simulação escolhido, os dados da simulação são salvos no InfluxDB, que é um banco de dados não relacional de série temporal. Na simulação com *delay*, o *timestamp* do dado é o UTC atual. Na simulação por número de ciclos, o primeiro *timestamp* é o horário UTC do início da simulação e a cada ciclo é adicionado um segundo.

3.2 Configuração da Linha de Envase

No PLSim, uma linha de envase pode conter até nove tipos de máquinas, que podem ser divididas em três grupos: o de máquinas gargalo[1], em inglês *bottleneck*, o de máquinas *upstream*, localizadas antes da máquina gargalo, e o de máquinas *downstream*, localizadas depois da máquina gargalo. A Figura 3.2 apresenta os tipos de máquinas e os três grupos mencionados estão identificados.

Figura 3.2 – Estrutura de uma linha de envase.

A Tabela 3.1 apresenta os nove tipos de máquinas e o valor máximo de máquinas do mesmo grupo que é possível habilitar na linha. Uma linha pode ter no máximo 31 máquinas, mesma estrutura de linha permitida no PaCo.

Tabela 3.1 – Grupo de máquinas.

Tipo da máquina	**Número máximo**
Third upstream machine	4
Second upstream machine	4
First upstream machine	4
Empty Container Inspector (ECI)	4
Filler	2
Pasteurizer	1
First downstream machine	4
Second downstream machine	4
Third downstream machine	4
Total	31

A configuração da linha de envase que se deseja usar na simulação é definida no arquivo `line_config`. Esse arquivo contém um dicionário (*dict* - estrutura em Python parecida com JSON) e uma lista, um para configuração das máquinas e um para os *buffers*, respectivamente. Os códigos apresentados nos Listings 3.1 e 3.2 mostram como essas variáveis estão organizadas.

As máquinas são separadas por grupos. É possível configurar o nome, a *flag* que indica se ela está habilitada ou não, a velocidade nominal, em CPM, e os parâmetros

[1] Máquina mais lenta da linha de envase

MTBF e MTTR (em segundos). Todas as 31 máquinas estão instanciadas e apenas aquelas que tem a *flag is_enabled* como *True* são levadas em consideração na simulação.

Listing 3.1 – Configuração das Máquinas.

```
machines_configuration = {
    "nome_do_grupo": [
        {
            "name": "maquina_1",
            "is_enabled": True,
            "nominal_speed": 20000,
            "type": 1,
            "mtbf": 0,
            "mttr": 0
        },
        ...
    ],
    ...
}
```

Já no caso dos *buffers* é possível configurar o nome e o tamanho máximo em unidades. Apenas os *buffers* que fazem parte da linha devem estar nessa lista. Os *buffers* ligam um grupo de máquinas ao outro.

Listing 3.2 – Configuração dos *Buffers*.

```
buffer_configuration = [
    {
        "name": "buffer_1",
        "size": 500
    },
    ...
]
```

Na Figura 3.2 é possível perceber que, se uma linha tem pelo menos uma máquina de cada grupo habilitada, são necessários dez *buffers*. O primeiro e o último *buffer* são considerados infinitos, pois eles devem ser capazes de armazenar o total de contêineres definidos na simulação. Portanto, nesses dois casos, o valor definido no arquivo de configurações não é respeitado.

A classe Line, quando instanciada, lê as configurações da linha definidas no ar-

quivo `line_config` e cria todos os seus objetos. É nesse momento também que as ligações entre máquinas e *buffers* é feita. Cada máquina tem a referência de seu *buffer* de entrada (*source buffer*) e de saída (*destination buffer*). E cada *buffer* tem duas listas, uma com as referências das máquinas que estão antes e outra com as máquinas que estão depois.

A lógica principal do simulador descreve o comportamento das máquinas, portanto está implementada na classe de *Machine*. Todas as máquinas começam a simulação no estado *Running* e, a cada ciclo a lógica é executada e o simulador define o estado de cada uma delas. Os estados possíveis, descritos na Seção 2.4.1, são: *Running*, *Starved*, *Blocked* e *Stopped*.

A Figura 3.3 é a representação simplificada da lógica principal do PLSim, que é executada para cada uma das máquinas a cada ciclo. A lógica é semelhante à apresentada para o BrewSim, com algumas pequenas diferenças. As variáveis das máquinas e *buffers* não eram lidas do servidor OPC UA e sim das variáveis de classe dos objetos. Foi necessário calcular, além da quantidade de contêineres disponíveis no *buffer* de entrada para cada máquina, e a quantidade de espaço disponível no *buffer* de saída para cada máquina. Ao final da lógica, as informações do ciclo são salvas no InfluxDB.

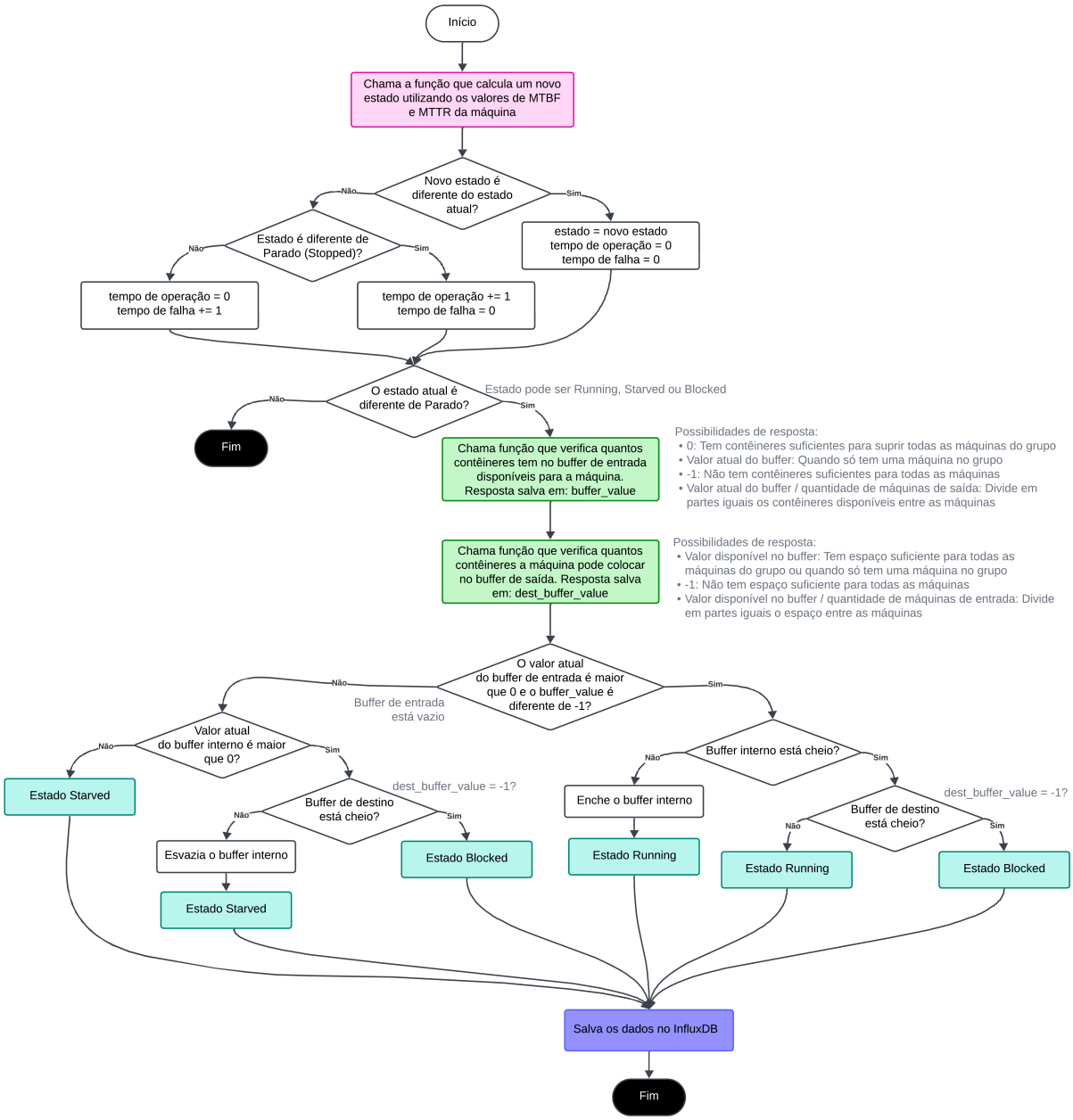

Figura 3.3 – Fluxograma simplificado que apresenta a lógica implementada no simulador PLSim.

3.3 Definição das Falhas Aleatórias no PLSim

Uma linha de envase real sofre com paradas de máquinas causadas por falhas, que podem ser problemas internos, como alguma peça quebrada, ou falta de algum material, por exemplo, falta de rótulos na máquina rotuladora. Com o objetivo de tornar o simulador mais próximo do funcionamento real de uma linha de envase, foi desenvolvida uma lógica para emular falhas no sistema.

O simulador BrewSim tinha a funcionalidade de falhas aleatórias implementada utilizando Cadeias de Markov, que, de forma simplificada, é uma técnica estatística que define uma matriz que contém a probabilidade de transição entre estados pré-definidos do sistema (SANTOS; ALENCAR, 2024). Como o intuito do PLSim é ser uma ferramenta fácil de ser utilizada, optou-se por não usar a Cadeia de Markov, já que a matriz de transição de estados não é tão trivial de se obter. Com isso, a estratégica foi desenvolver um sistema de configuração simplificada, mas que traria o mesmo benefício para o simulador: que é implementar as paradas aleatórias das máquinas. Para isso, foram utilizados dois parâmetros, MTBF e MTTR, cujo conceito foi apresentado na Subseção 2.4.2.

O MTBF foi utilizado para determinar o ciclo de início da parada, já o MTTR foi utilizado para determinar o ciclo de fim da parada. Para que as paradas não sejam ritmadas, ou seja, que aconteçam sempre no mesmo intervalo, foi utilizada uma variação de ±10% do valor do MTBF para determinar o ciclo de início delas. A mesma variação foi aplicada na duração das paradas, onde o fim da parada é determinado pelo ciclo de início mais o MTTR com a varição de 10%. Além disso, com o intuito de adicionar uma aleatoriedade no sistema, há 50% de chance da parada acontecer, isso porque em todo ciclo de início de parada é sorteado um número aleatório de 1 a 10 e, apenas se esse valor for maior que 5, a parada será aplicada no sistema.

Foram criadas duas variáveis, *cycle_failure* e *cycle_recovery* para indicar qual o ciclo para iniciar e encerrar a falha, respectivamente. Quando a máquina está em operação, a variável *cycle_failure* contém o valor do ciclo em que a falha deve ser iniciada e a variável *cycle_recovery* recebe o valor -1. Porém, quando a máquina está parada, a variável *cycle_failure* recebe o valor -1 e a variável *cycle_recovery* recebe o valor do ciclo em que a parada deve ser finalizada. Portanto, sempre uma das variáveis é -1 e a outra contém um valor de um ciclo futuro.

No início do código, a variável *cycle_failure* recebe o valor do MTBF ±10% do MTBF e a variável *cycle_recovery* recebe -1. Caso o valor do MTBF seja 0, a variável *cycle_failure* também recebe o valor -1. O código apresentado no Listing 3.3 representa como é feita a inicialização das variáveis.

Listing 3.3 – Inicialização das variáveis *cycle_failure* e *cycle_recovery*.

```
se mtbf diferente de 0 então
    limite = 10% de mtbf
    intervalo = número inteiro aleatório entre [-limite, +limite]
    ciclo de falha = mtbf + intervalo
senão
    ciclo de falha = -1
```

```
ciclo de recuperação = -1
```

O código do Listing 3.4 ilustra o método que contém a lógica de parada da máquina na simulação, onde o parâmetro de entrada *i* é o número do ciclo. Primeiro o código verifica se está no ciclo definido na variável *cycle_failure*, se sim, verifica se a máquina está no estado *Running*. Se estiver no estado *Running*, um número inteiro de 1 a 10 é sorteado para decidir se a máquina será parada ou não. Se o número for maior que 5, o estado da máquina é alterado para *Stopped* e o valor do ciclo que a parada irá encerrar é calculado e salvo na variável *cycle_recovery*. Além disso, a variável *cycle_failure* recebe valor -1 e a variável lógica *stop_forced* é definida como verdadeira. Essa variável é utilizada para garantir que a máquina está em uma parada forçada, induzida pela simulação. Se o número sorteado for menor que 5 ou a máquina não estiver no estado *Running*, a variável lógica *next_fault* é alterada de falsa para verdadeira. Em seguida, se a variável *next_fault* for verdadeira, significa que a falha não foi induzida e um novo valor para a variável *cycle_failure* é calculado.

Por fim, o código verifica se o número do ciclo é o mesmo definido na variável *cycle_recovery*. Se sim, há uma nova condição no código que confere se o estado da máquina é diferente de *Running* e se a variável lógica *stop_forced* é verdadeira, para garantir que a máquina esteja parada e que o motivo foi a parada forçada pelo simulador. Se sim, o estado da máquina é alterado para *Running* e a variável lógica *stop_forced* é alterada para falsa. Em seguida, o valor do ciclo que uma nova parada deve ser iniciada é calculado e salvo na variável *cycle_failure*.

O método retorna a variável *state*, que é inicializada com o valor do estado atual da máquina. Isso significa que, caso nenhuma das condições descritas anteriormente sejam atendidas, a máquina permanece no mesmo estado em que já se encontrava.

Listing 3.4 – Método que inicia e encerra as falhas das máquinas.

```
variáveis de entrada
i: número do ciclo

# Obtém valor do estado atual da máquina
estado = estado atual da máquina

# Verifica se está no ciclo de criar uma falha
se i == ciclo de falha então
    se o estado == 'Running' então
```

```
        decisão de parada = número inteiro aleatório entre [1, 10]
        se decisão de parada > 5 então:
            estado = 'Stopped'

            limite = 10% de mttr
            intervalo = número inteiro aleatório entre [-limite,
            ↪  +limite]
            ciclo de recuperação = mttr + intervalo

            ciclo de falha = -1
            parada forçada = verdadeiro

        senão:
        próxima falha = verdadeiro

    se próxima falha é verdadeiro então:
        limite = 10% de mtbf
        intervalo = número inteiro aleatório entre [-limite, +limite]
        ciclo de falha = mtbf + intervalo

# Verifica se está no ciclo encerrar a falha
se i == ciclo de recuperação então:
    se estado é diferente de 'Running' e para forçada é verdadeiro então
        estado = 'Running'
        parada forçada = falso

    limite = 10% de mtbf
    intervalo = número inteiro aleatório entre [-limite, +limite]
    ciclo de falha = mtbf + intervalo

    ciclo de recuperação = -1

retorna estado
```

3.4 Modelagem dos *Buffers*

No simulador BrewSim os *buffers* são representados como contadores, ou seja, a capacidade máquina e a acumulação atual são definidas em contêineres. Como o PLSim

utilizou a mesma lógica implementada no BrewSim, trata os *buffers* da mesma forma. Para uma simulação de uma linha de envase genérica, essa abordagem é suficiente, já que ela consegue representar bem o comportamento dos *buffers*, porém, quando se deseja representar uma linha de envase real, não é a maneira mais adequada. Isso porque o tamanho máximo dos *buffers* não é definido em contêineres, existem sensores em pontos estratégicos que são dizer se o *buffer* está cheio ou vazio.

A modelagem correta dos *buffers* é um desafio, é necessário um estudo para identificar a melhor solução para este problema. No passado, a empresa já tentou algumas iniciativas, incluindo até processamento de imagem, mas nenhuma delas foi levada adiante. Para alcançar esse objetivo, tem-se algumas variáveis disponíveis para serem utilizadas na modelagem, como por exemplo:

- Contador de contêineres das máquinas;
- Sensores ao longo do *buffer*, que quando acionados dizem que há contêineres passando por eles;
- Dimensões dos *buffers*, largura e comprimento.

4 Resultados do Simulador PLSim

Este capítulo apresenta os testes realizados no simulador PLSim para validar o seu funcionamento. Ainda durante o desenvolvimento do PLSim, alguns testes foram executados a fim de garantir que o código estava funcional, ou seja, não continha erros de sintaxe. Em seguida, foram executados testes para validar a lógica da simulação. A Seção 4.1 apresenta os testes de validação realizados considerando uma linha simples, a Seção 4.2 apresenta os testes de validação das ramificações na linha e a Seção 4.3 apresenta os testes das falhas aleatórias.

A lista a seguir apresenta os parâmetros que devem ser configurados para a execução de cada teste no PLSim.

- *name*: Nome do teste
- *n*: Quantidade de ciclos, caso esse tipo de simulação seja escolhida
- *target*: Quantidade total de contêineres a serem processados pela linha
- *machines_configuration* > *name*: Nome da máquina
- *machines_configuration* > *is_enabled*: Flag que habilita/desabilita cada uma das máquinas
- *machines_configuration* > *nominal_speed*: Velocidade nominal em CPM (*containers per minute*)
- *machines_configuration* > *mtbf*: Tempo médio entre falhas em segundos
- *machines_configuration* > *mttr*: Tempo médio de reparo em segundos
- *buffer_condigurarion* > *name*: Nome do *buffer*.
- *buffer_condigurarion* > *size*: Tamanho máximo de cada *buffer*.

Todos os testes foram realizados no modo de simulação em que se define o número de ciclos desejado. Os dados da simulação foram salvos no InfluxDB, que foi executado utilizando um *container Docker*. O *timestamp* dos dados foi salvo no fuso-horário UTC, onde o primeiro valor é a data e hora exata em que a simulação foi iniciada e, a cada ciclo, um segundo é adicionado. Os testes foram executados em um notebook com as seguintes especificações:

- Processador: 11th Gen Intel(R) i7-11390H @ 3.40GHz 2.92GHz;
- Memória RAM: 16GB;
- Sistema operacional: Windows 11.

Nas seções seguintes são apresentados dois tipos de gráficos, um para ilustrar o estado das máquinas ao longo do tempo de simulação e outro para ilustrar o contador de

contêineres dos *buffers* da linha. Em ambos os casos, o eixo X é o tempo de simulação convertido para o horário UTC-03:00, que é o horário de Brasília, sendo o primeiro valor o momento em que a simulação foi iniciada. No primeiro tipo de gráfico de cada teste, o eixo Y contém uma linha do tempo para cada máquina habilitada, que apresenta qual estado a máquina assumiu ao longo da simulação. Os estados podem ser identificados pelo escrito ou pelas cores, sendo verde para o estado *Running*, roxo para *Starved*, azul para *Blocked* e vermelho pra *Stopped*. Já nos gráficos dos *buffers*, o eixo Y apresenta o valor do contador de contêineres, e apresenta como a quantidade de contêineres variou em cada *buffer* ao longo da simulação.

4.1 Fase 1: Linhas Simples

Na primeira fase de validação do PLSim, foram definidos oito cenários de teste e, em todos, apenas uma máquina de cada grupo estava habilitada, totalizando nove máquinas em cada linha. Como o objetivo dessa etapa é validar a lógica implementada, a ideia é variar a velocidade de uma ou mais máquinas e verificar o comportamento da linha, analisando o estado das máquinas e acumulação nos *buffers*.

As Figuras 4.1 e 4.2 apresentam os cenários de teste propostos nesta fase. No Teste 1 a velocidade de todas as máquinas foi definida como 10k CPM e o tamanho máximo dos *buffers* como 1000 contêineres. Essa configuração serviu como base para todos os outros cenários, incluindo os testes das Fases 2 e 3. No Teste 2 e 3 todas as máquinas foram aceleradas, para 20k CPM, e desaceleradas, para 5k CPM, respectivamente. Os Testes 4 e 5 alteram a velocidade apenas da máquina gargalo, identificada como *Filler* nas figuras. Já os Testes 6 e 7 alteram a velocidade apenas da primeira máquina da linha. O Teste 8 tinha o objetivo de estressar um pouco mais o sistema, para que fosse possível perceber a dinâmica da máquina alternando entre os estados*Running*, *Starved* e *Blocked*. Para isso, o tamanho máximo dos *buffers* foi reduzido para 500 contêineres e a velocidade de todas as máquina alterada para 20k CPM.

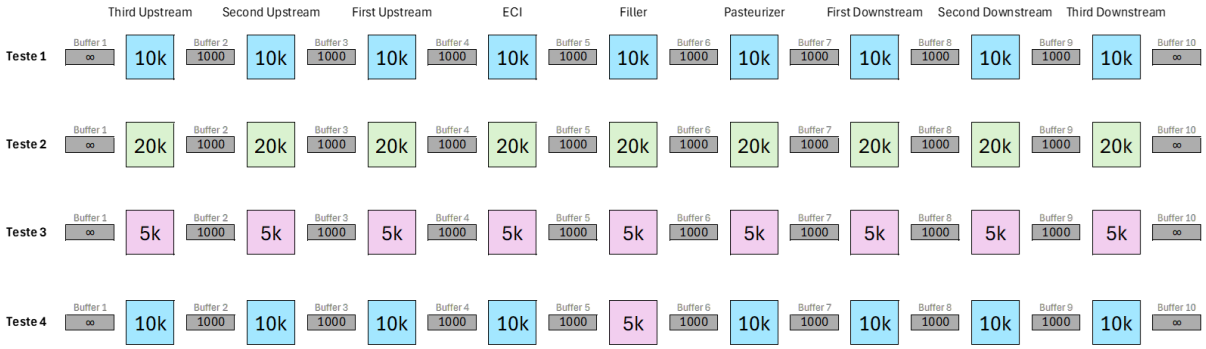

Figura 4.1 – Representação da linha de envase configurada nos Testes 1 a 4.

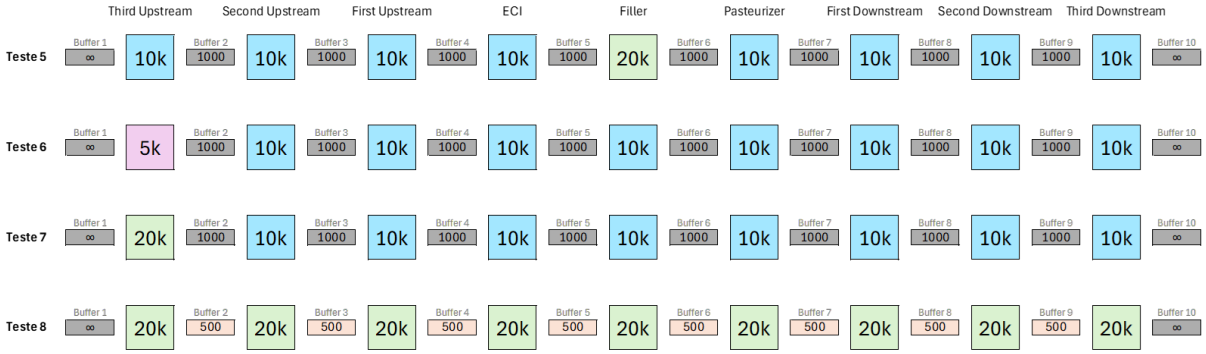

Figura 4.2 – Representação da linha de envase configurada nos Testes 5 a 8.

Após a definição dos oito cenários de testes, todas as simulações foram executadas, e em seguida, os resultados foram analisados para validar a lógica de simulação. Nesta etapa, os valores dos parâmetros MTBF e MTTR foram definidos como zero, com isso não foi necessário executar o mesmo cenário mais de uma vez, já que o resultado seria sempre o mesmo, pois as simulações são determinísticas. Após a detecção de inconsistências nos resultados, os erros encontrados foram corrigidos e novos testes foram feitos, conforme será apresentado nesta seção.

4.1.1 Fase 1: Linhas simples - Teste 1

O primeiro teste considera uma linha simples, em que todas as máquinas operam na mesma velocidade. As tabelas 4.1 e 4.2 apresentam as configurações das máquinas e *buffers*, respectivamente. Todos os testes seguintes usam este primeiro teste como base.

Tabela 4.1 – Configuração das máquinas - Teste 1.

Grupo de máquinas	**Quantidade de máquinas habilitadas**	**Tamanho *buffer* interno**	**Velocidade nominal (CPM)**
Third upstream machine	1	1	10000
Second upstream machine	1	1	10000
First upstream machine	1	1	10000
Empty Container Inspector (ECI)	1	1	10000
Filler	1	1	10000
Pasteurizer	1	1	10000
First downstream machine	1	1	10000
Second downstream machine	1	1	10000
Third downstream machine	1	1	10000

Tabela 4.2 – Configuração dos *buffers* - Teste 1.

Nome do *buffer*	**Tamanho máximo**
begin_to_third_upstream	1000
third_to_second_upstream	1000
second_to_first_upstream	1000
first_upstream_to_eci	1000
eci_to_filler	1000
filler_to_pasteurizer	1000
pasteurizer_to_first_downstream	1000
first_to_second_downstream	1000
second_to_third_downstream	1000
third_downstream_to_end	1000

A Tabela 4.3 apresenta os dados de configuração deste teste.

Tabela 4.3 – Configuração da Simulação 1.

Número de ciclos (n)	10000
Unidades de entrada (*target*)	1000000

O número de ciclos foi suficiente para processar por completo todas as unidades de entrada. Cada máquina fica 6026 ciclos no estado *Running*, e em seguida vai para o estado *Starved*. O processamento total se deu em 6050 ciclos (equivalente a mais de 1 hora e quarenta minutos de funcionamento da linha). No restante do tempo, a linha permanece em repouso: todas as máquinas no estado *Starved* e todos os *buffers* vazios, exceto o último.

Como nesta simulação a velocidade nominal de todas as máquinas era a mesma, a linha entra em equilíbrio, ou seja, todos os *buffers* ficam com a mesma quantidade de unidades durante o período estacionário. Isso acontece porque a quantidade de unidades que a máquina anterior transfere por segundo é a mesma que a máquina seguinte retira do *buffer*. Para a velocidade de 10k CPM, os *buffers* se estabilizam com 497 itens, que correspondem a acumulação de 49,7%. A seguir está a descrição do que acontece nos primeiros ciclos da simulação, que explica o número 497 de itens nos *buffers* e como é feita a inicialização das máquinas.

- Ciclo 0: Todas as máquinas iniciam a simulação no estado *Running*.
- Ciclo 1: Todas as máquinas mudam o estado para *Starved*, exceto a primeira máquina, third_upstream_machine_1, que permanece no estado *Running*. Esta máquina enche seu *buffer* interno: $counter = 1$.
- Ciclo 2: Apenas a primeira máquina, third_upstream_machine_1, no estado *Running*. O *buffer* interno da máquina já está cheio, então 166 unidades são transferidas para o *buffer* de destino: third_to_second_upstream. O número de unidades é obtido através do cálculo apresentado na Equação 4.1, que a faz conversão da velocidade de CPM, contêineres por minuto, para CPS, contêineres por segundo, que é a quantidade de contêineres que a máquina processa por ciclo.

$$10000CPM/60s = 166CPS \tag{4.1}$$

- Ciclo 3: Apenas primeira máquina, third_upstream_machine_1, no estado *Running*. Mais 166 unidades são transferidas para o *buffer* third_to_second_upstream, totalizando 332.
- Ciclo 4: Primeira e segunda máquinas, third_upstream_machine_1 e second_upstream_machine_1, no estado *Running*. Enche o *buffer* interno da segunda máquina. Mais 166 unidades são transferidas para o *buffer* third_to_second_upstream, totalizando 497 unidades ($166 * 3 - 1 = 497$).
- Ciclo 5: Primeira e segunda máquinas, third_upstream_machine_1 e second_upstream_machine_1, no estado *Running*. Primeira máquina transfere mais 166 unidades para o *buffer* third_to_second_upstream. A segunda máquina transfere 166 unidades do *buffer* third_to_second_upstream para o *buffer* second_to_first_upstream.

A Figura 4.3 apresenta a dinâmica da inicialização das máquinas e dos contadores dos *buffers*. É possível perceber que a alteração do estado de máquina segue um padrão. O mesmo acontece com os *buffers*, em que os ciclos descritos nesta seção se repetem até que todas as máquinas sejam inicializadas.

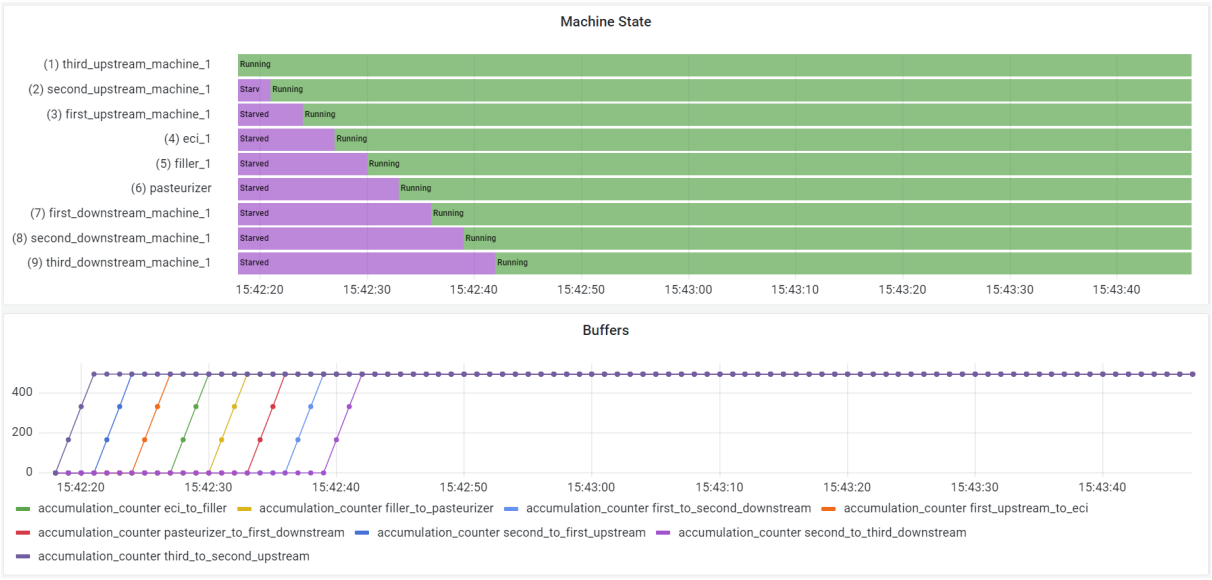

Figura 4.3 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 1.

A Figura 4.4 apresenta a visão geral dos *buffers* em todo o período do teste. O primeiro gráfico contém a dinâmica dos *buffers* que ficam entre as máquinas. O segundo gráfico apresenta a dinâmica do primeiro e do último *buffer*, que são os considerados infinitos.

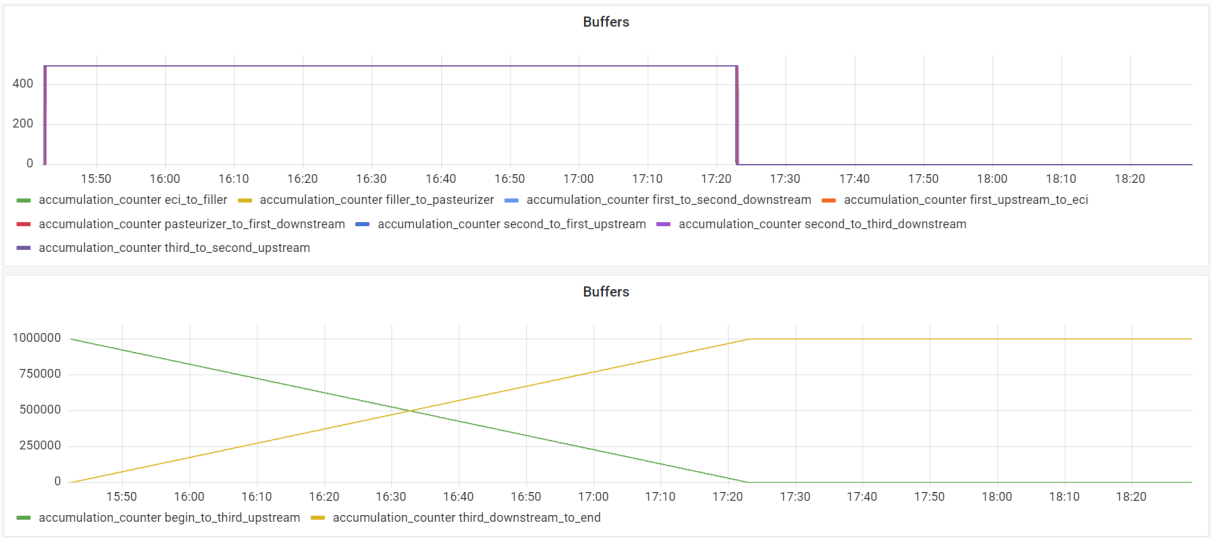

Figura 4.4 – Gráfico com o comportamento dos *buffers* durante todo o Teste 1.

4.1.2 Fase 1: Linhas simples - Teste 2

Para o segundo teste, a configuração dos *buffers* e da simulação permaneceram iguais as do teste anterior, apresentadas nas tabelas 4.2 e 4.3. Foram alteradas as configurações das máquinas, conforme apresenta a Tabela 4.4, onde a velocidade de todas as máquinas foi alterada de 10k CPM para 20k CPM.

Tabela 4.4 – Configuração das máquinas - Teste 2.

Grupo de máquinas	Quantidade de máquinas habilitadas	Tamanho *buffer* interno	Velocidade nominal (CPM)
Third upstream machine	1	1	20000
Second upstream machine	1	1	20000
First upstream machine	1	1	20000
Empty Container Inspector (ECI)	1	1	20000
Filler	1	1	20000
Pasteurizer	1	1	20000
First downstream machine	1	1	20000
Second downstream machine	1	1	20000
Third downstream machine	1	1	20000

O objetivo principal do teste era observar o comportamento da linha para a nova velocidade das máquinas. Neste teste, por ciclo, cada máquina movimenta 333 unidades.

$$20000CPM/60s = 333CPS \tag{4.2}$$

Diferente do Teste 1, houve um período transitório no início da simulação, em que o número de unidades em cada *buffer* variou entre 667 e 998. Já no regime estacionário, os *buffers* estabilizaram com 667 unidades. O comportamento descrito pode ser observado na Figura 4.5.

Figura 4.5 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 2.

Cada máquina fica 3009 ciclos no estado *Running*, e em seguida vai para o estado *Starved*. As 1000000 unidades demoram 3029 ciclos para serem processadas. A Figura 4.6 apresenta a visão geral dos *buffers* em todo o período do teste.

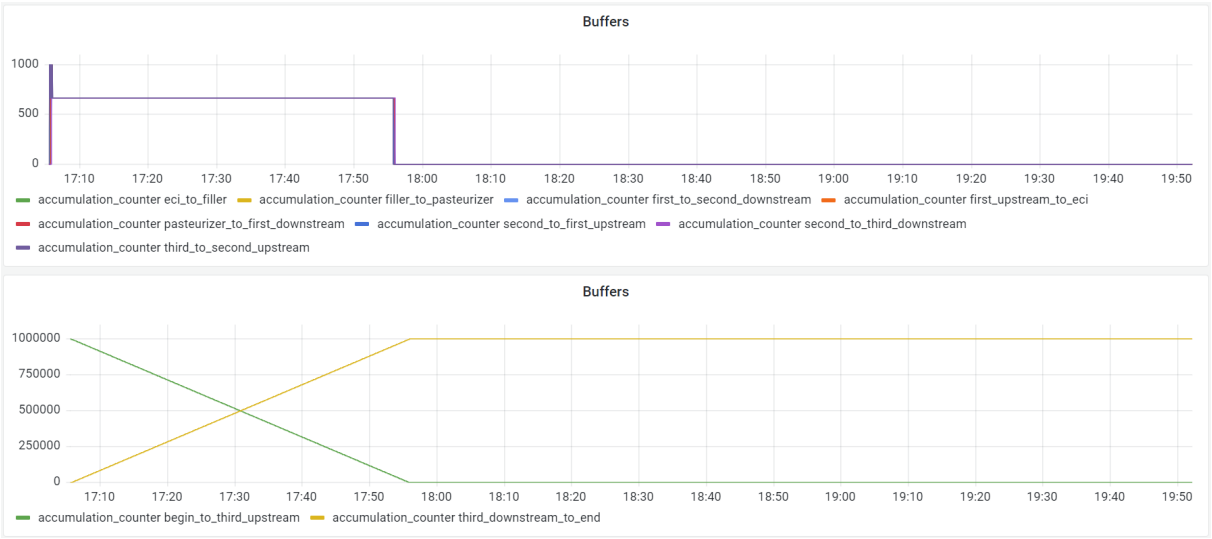

Figura 4.6 – Gráfico com o comportamento dos *buffers* durante todo o Teste 2.

4.1.3 Fase 1: Linhas simples - Teste 3

O terceiro teste é semelhante ao Teste 2, mas, ao invés de acelerar todas as máquinas em relação ao Teste 1, elas foram desaceleradas. As configurações dos *buffers* e da simulação permaneceram iguais, apresentadas nas Tabelas 4.2 e 4.3. Foram alteradas as configurações das máquinas, conforme apresenta a Tabela 4.5, onde a velocidade de todas as máquinas foi alterada de 10k CPM para 5k CPM.

Tabela 4.5 – Configuração das máquinas - Teste 3.

Grupo de máquinas	Quantidade de máquinas habilitadas	Tamanho *buffer* interno	Velocidade nominal (CPM)
Third upstream machine	1	1	5000
Second upstream machine	1	1	5000
First upstream machine	1	1	5000
Empty Container Inspector (ECI)	1	1	5000
Filler	1	1	5000
Pasteurizer	1	1	5000
First downstream machine	1	1	5000
Second downstream machine	1	1	5000
Third downstream machine	1	1	5000

Neste teste, os *buffers* estabilizaram com 248 unidades e o comportamento é semelhante ao descrito no Teste 1. Cada máquina transfere 83 unidades por ciclo, conforme apresentado na Equação 4.3. O valor de unidades que permanecem nos *buffers* pode ser encontrado através do cálculo: $83 * 3 - 1 = 248$. O comportamento descrito pode ser observado na Figura 4.7.

$$5000CPM/60s = 83CPS \tag{4.3}$$

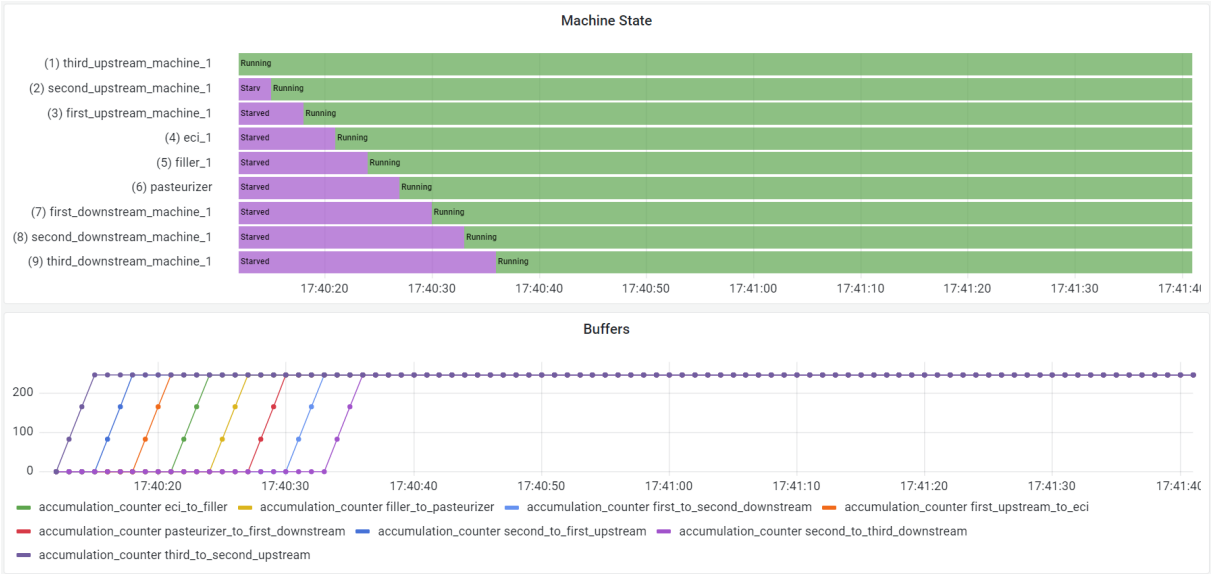

Figura 4.7 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 3.

Os 10000 ciclos não foram suficientes para processar todas as unidades de entrada. A Tabela 4.6 apresenta o estado de cada um dos *buffers* no último ciclo e a Figura 4.8 apresenta a visão geral dos *buffers* em todo o período do teste.

Tabela 4.6 – Estado dos *buffers* no último ciclo - Teste 3.

Nome do *buffer*	**Contador**
begin_to_third_upstream	170165
third_to_second_upstream	248
second_to_first_upstream	248
first_upstream_to_eci	248
eci_to_filler	248
filler_to_pasteurizer	248
pasteurizer_to_first_downstream	248
first_to_second_downstream	248
second_to_third_downstream	248
third_downstream_to_end	827842
Buffer interno das máquinas	9
Total	1000000

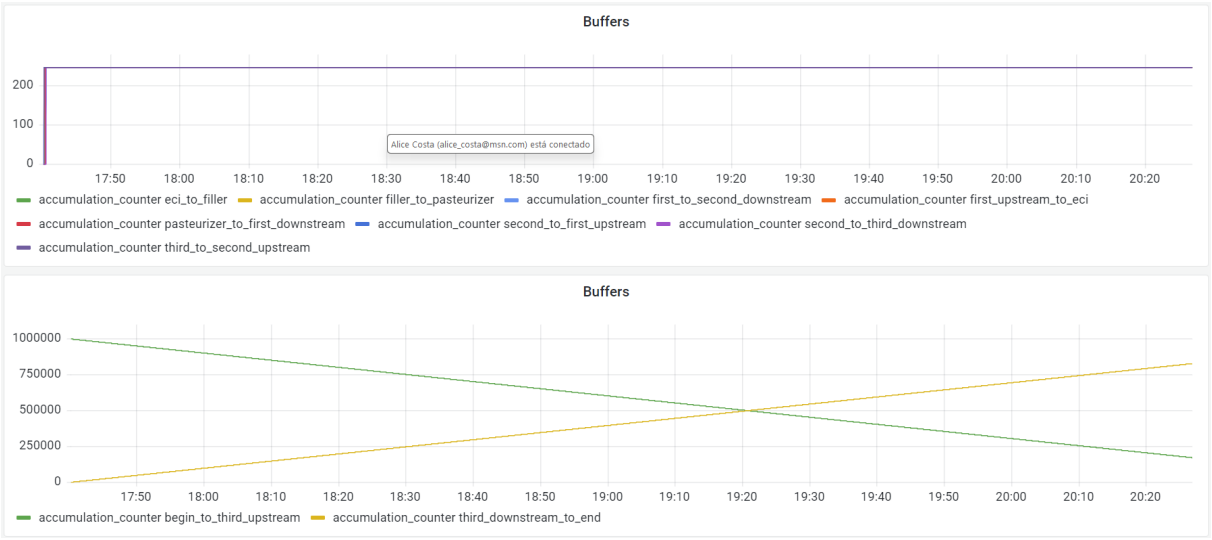

Figura 4.8 – Gráfico com o comportamento dos *buffers* durante todo o Teste 3.

4.1.4 Fase 1: Linhas simples - Teste 4

Neste teste, a configuração dos *buffers* e da simulação também permaneceram iguais as do Teste 1, apresentadas nas Tabelas 4.2 e 4.3. O diferencial foi na configuração das máquinas, em que todas têm velocidade de 10k CPM exceto a máquina "gargalo"(*Filler*), que tem velocidade de 5k CPM, conforme apresenta a Tabela 4.7.

Tabela 4.7 – Configuração das máquinas - Teste 4.

Grupo de máquinas	**Quantidade de máquinas habilitadas**	**Tamanho *buffer* interno**	**Velocidade nominal (CPM)**
Third upstream machine	1	1	10000
Second upstream machine	1	1	10000
First upstream machine	1	1	10000
Empty Container Inspector (ECI)	1	1	10000
Filler	1	1	5000
Pasteurizer	1	1	10000
First downstream machine	1	1	10000
Second downstream machine	1	1	10000
Third downstream machine	1	1	10000

Nesse caso, os 10000 ciclos não foram suficientes para processar todas as unidades e, na construção da tabela com o estado de cada um dos *buffers* no último ciclo foi possível perceber um erro. Como mostra a segunda coluna da Tabela 4.8, 660 unidades desapareceram da linha. Após uma investigação, o erro foi encontrado no código e o teste foi refeito, a terceira coluna da tabela apresenta os resultados atualizados.

Tabela 4.8 – Estado dos *buffers* no último ciclo - Teste 4.

Nome do *buffer*	**Contador com erro**	**Contador sem erro**	**Diferença**
begin_to_third_upstream	167489	169481	+ 1992
third_to_second_upstream	917	917	0
second_to_first_upstream	917	917	0
first_upstream_to_eci	917	917	0
eci_to_filler	917	917	0
filler_to_pasteurizer	83	83	0
pasteurizer_to_first_downstream	83	83	0
first_to_second_downstream	83	83	0
second_to_third_downstream	83	83	0
third_downstream_to_end	827842	826510	- 1332
Buffer interno das máquinas	9	9	0
Total	999340	1000000	-
Faltam	**660**	**0**	**-**

Esse erro na lógica foi herdado do simulador utilizado como base e colocar uma máquina com a velocidade exatamente duas vezes menor que as outras foi fundamental para detectar essa inconsistência. O PLSIM calcula o número de unidades que a máquina processa a cada ciclo e esse valor depende de algumas condições, como por exemplo, se o *buffer* interno da máquina já está cheio ou não ou se tem unidades suficientes no *buffer* de entrada, além de parâmetros como a velocidade nominal da máquina. Existe no código uma condição que verifica se a quantidade de unidades disponíveis no *buffer* de entrada é maior que a velocidade da máquina. Essa condição serve para verificar se a máquina pode processar 100% da sua capacidade ou um pouco menos. Se a comparação for verdadeira, os comandos da condição *if* são executados, e se for falsa, o *else* é executado. No caso deste teste, aconteceu a situação dos valores serem iguais, pois a velocidade do *filler* era metade da velocidade do restante da linha, o que fez com que a lógica executasse os comando da condição *else*, que geravam um valor errado e as unidades sumiam da linha. Para solucionar esta situação o sinal de menor foi substituído pelo de menor igual, dessa forma a comparação dava verdadeiro, o *if* era executado e o valor era calculado corretamente. Essa condição aparece duas vezes no código e foi corrigida nas duas situações.

Neste teste foi possível perceber um comportamento transitório na inicialização da linha, como mostra a Figura 4.9. Quando o sistema entra no estado estacionário, os *buffers* que estão antes do filler, que é a máquina mais lenta, estabilizam com 917 unidades, enquanto os que estão depois estabilizam com 83 unidades, como mostra a Tabela 4.8 e a Figura 4.10

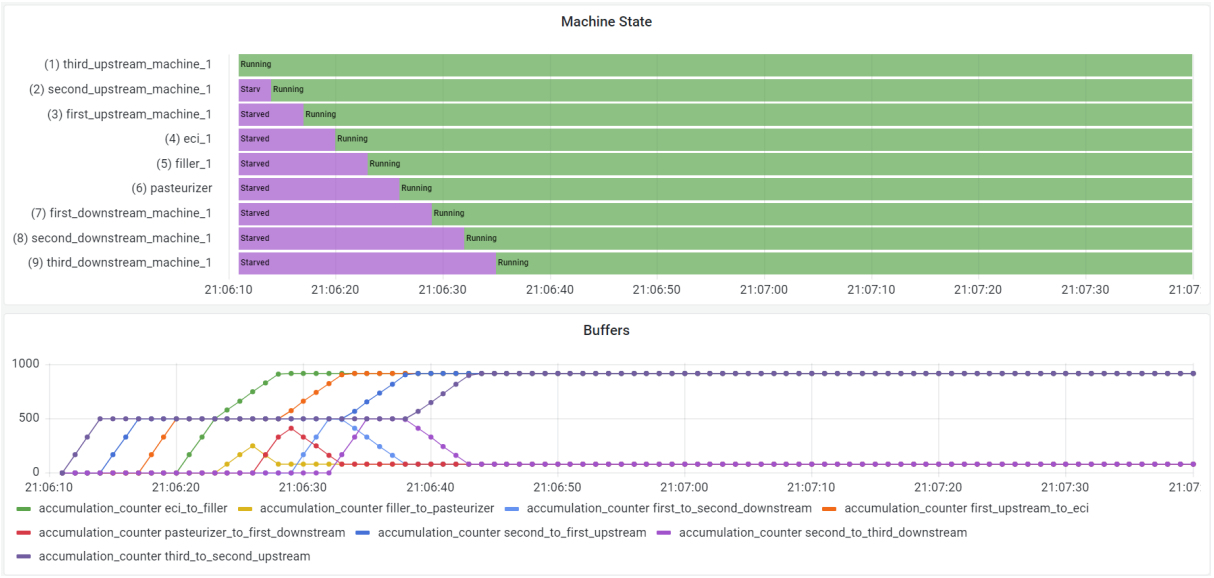

Figura 4.9 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 4.

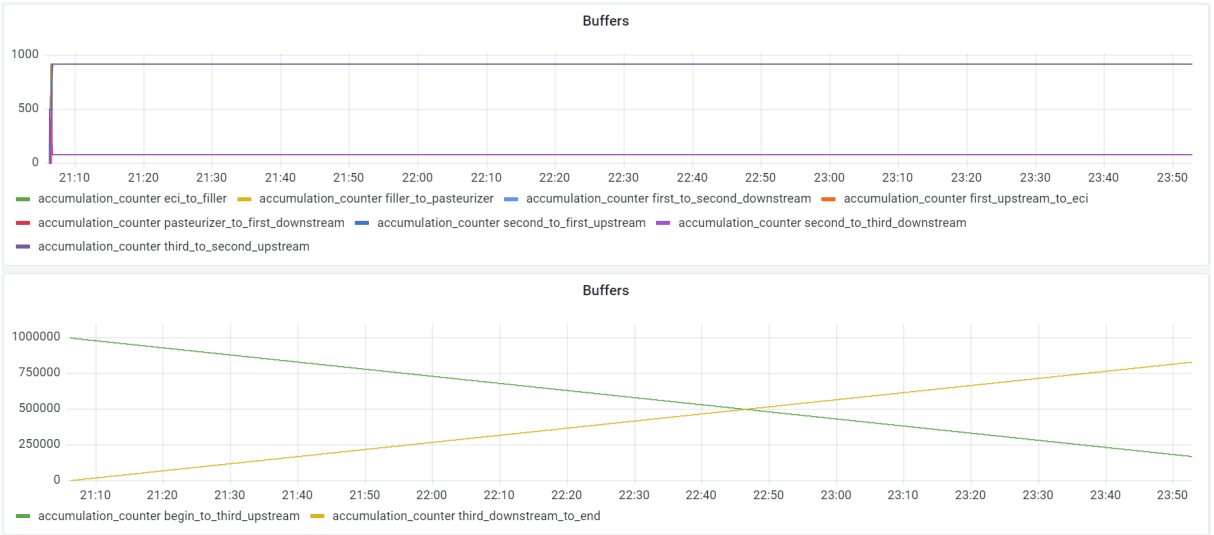

Figura 4.10 – Gráfico com o comportamento dos *buffers* durante todo o Teste 4.

4.1.5 Fase 1: Linhas simples - Teste 5

Este teste, assim como o Teste 4 altera a velocidade de apenas uma máquina, mas desta vez ela é mais rápida que o restante da linha. A configuração dos *buffers* e da simulação permaneceram iguais as do Teste 1, apresentadas nas Tabelas 4.2 e 4.3. Desta vez, o *filler* foi configurado com velocidade de 20k CPM, conforme apresenta a Tabela 4.9.

Tabela 4.9 – Configuração das máquinas - Teste 5.

Grupo de máquinas	**Quantidade de máquinas habilitadas**	**Tamanho *buffer* interno**	**Velocidade nominal (CPM)**
Third upstream machine	1	1	10000
Second upstream machine	1	1	10000
First upstream machine	1	1	10000
Empty Container Inspector (ECI)	1	1	10000
Filler	1	1	20000
Pasteurizer	1	1	10000
First downstream machine	1	1	10000
Second downstream machine	1	1	10000
Third downstream machine	1	1	10000

Neste caso, os 10000 ciclos foram suficientes para processar todas as unidades de entrada. Como o teste anterior apresentou erros, sentiu-se a necessidade de uma análise mais detalhada para ter certeza do funcionamento correto. Durante a verificação dos dados, outro erro foi encontrado. No 13° ciclo, na inicialização da máquina mais rápida, 331 unidades sumiam da linha e voltavam durante a finalização do processamento das unidades. Assim como no caso anterior, o erro foi encontrado e corrigido. A Tabela 4.10 apresenta uma comparação do contador dos *buffers* no 26° ciclo de cada uma das simulações, que é quando todas as máquinas já estão inicializadas.

Tabela 4.10 – Estado dos *buffers* no último ciclo - Teste 5.

Nome do *buffer*	**Contador com erro**	**Contador sem erro**	**Diferença**
begin_to_third_upstream	995849	995849	0
third_to_second_upstream	497	497	0
second_to_first_upstream	497	497	0
first_upstream_to_eci	497	497	0
eci_to_filler	166	166	0
filler_to_pasteurizer	497	828	+ 331
pasteurizer_to_first_downstream	497	497	0
first_to_second_downstream	497	497	0
second_to_third_downstream	497	497	0
third_downstream_to_end	166	166	0
Buffer interno das máquinas	9	9	0
Total	999669	1000000	-
Faltam	**331**	**0**	**-**

Após a correção do erro descrito no Teste 4, o Teste 5 foi refeito a fim de conferir se a inconsistência dos dados ocorreu pelo mesmo erro no código. Analisando os dados foi possível perceber que a inconsistência persistia e outra investigação foi feita para encontrar a causa deste segundo erro.

O segundo erro também foi herdado do simulador base e causa foi detectada na mesma condição do código descrita no Teste 4, que verifica se a quantidade de unidades disponíveis no *buffer* de entrada é maior que a velocidade da máquina. Essa etapa acontece dentro do bloco *if* e do bloco *else* de uma condição que verifica se o contador do *buffer*

interno da máquina é menor que o valor máximo do *buffer* interno (bloco *if*) ou não (bloco else). Então tem-se duas situações:

1. *Buffer* interno da máquina não está 100% cheio e a velocidade nominal da máquina é maior que a quantidade de unidades disponíveis.
2. *Buffer* interno da máquina está 100% cheio e a velocidade nominal da máquina é maior que a quantidade de unidades disponíveis.

Para a situação descrita em 1 foi necessário limitar o valor processado no ciclo à quantidade disponível no *buffer* interno e para a situação 2 foi necessário limitar à quantidade disponível no *buffer* de saída (*destination buffer*). Na inicialização, as unidades que sumiram dos *buffers* da linha estavam no *buffer* interno da máquina mais rápida, que só tinha capacidade para 1 unidade e recebeu 332. Por esse motivo, durante a finalizações elas reapareciam na linha.

Foi possível perceber um comportamento transitório durante a inicialização da linha no *buffer* que está localizado imediatamente antes da máquina mais rápida: eci_to_filler. A Figura 4.11 apresenta o comportamento das máquinas e dos bufffers na inicialização do teste. Depois que o sistema entra no estado estacionário, apenas dois *buffers* não estabilizaram com 497 unidades, sendo eles os *buffers* antes e depois da máquinas mais rápida. O *buffer* eci_to_filler estabilizou com 166 unidades e o *buffer* filler_to_pasteurizer estabiliza em 828 unidades, isso porque o filler processa as unidades duas vezes mais rápido que o restante das máquinas. Esse comportamento pode ser observado tanto no gráfico da Figura 4.11 quanto no gráfico da Figura 4.12.

Figura 4.11 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 5.

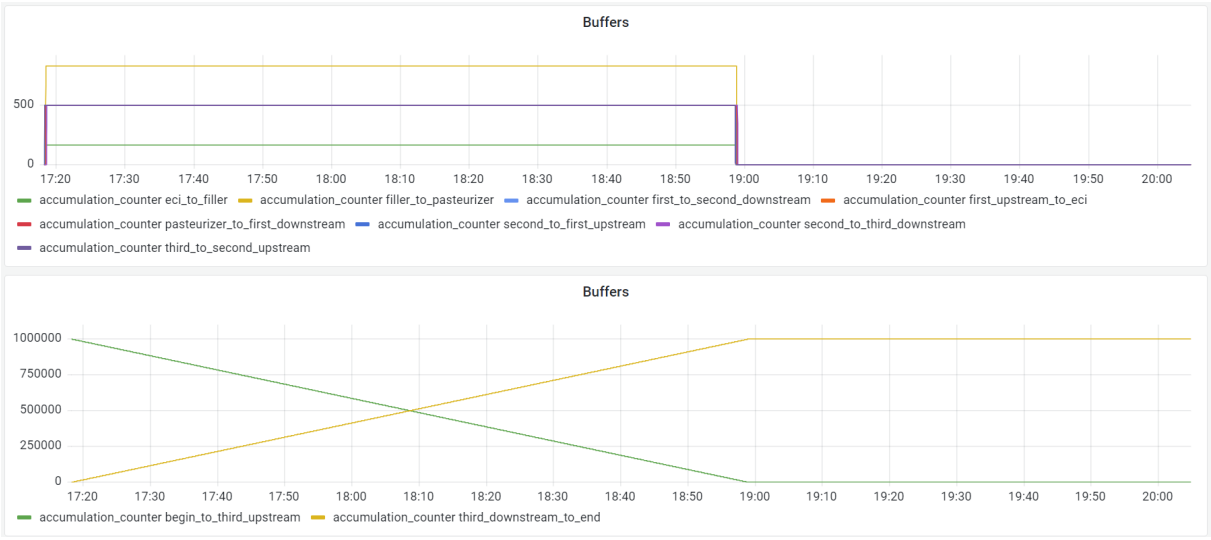

Figura 4.12 – Gráfico com o comportamento dos *buffers* durante todo o Teste 5.

4.1.6 Fase 1: Linhas simples - Teste 6

Este teste é semelhante ao Teste 4, porém, ao invés de alterar a velocidade do *filler*, a velocidade da primeira máquina foi alterada de 10k CPM para 5k CPM. A Tabela 4.11 apresenta a configuração das máquinas.

Tabela 4.11 – Configuração das máquinas - Teste 6.

Grupo de máquinas	**Quantidade de máquinas habilitadas**	**Tamanho *buffer* interno**	**Velocidade nominal (CPM)**
Third upstream machine	1	1	5000
Second upstream machine	1	1	10000
First upstream machine	1	1	10000
Empty Container Inspector (ECI)	1	1	10000
Filler	1	1	10000
Pasteurizer	1	1	10000
First downstream machine	1	1	10000
Second downstream machine	1	1	10000
Third downstream machine	1	1	10000

Os 10000 ciclos não foram suficientes para processar todas as unidades. Como descrito no Teste 4, foi encontrado um erro no código. Como todos os testes foram executados antes da análise dos dados, foram feitas duas simulações: uma com erro e a outra sem. A Tabela 4.12 contém o estado dos *buffers* no último ciclo da simulação.

Tabela 4.12 – Estado dos *buffers* no último ciclo - Teste 6.

Nome do *buffer*	**Contador com erro**	**Contador sem erro**	**Diferença**
begin_to_third_upstream	170165	169481	0
third_to_second_upstream	83	83	0
second_to_first_upstream	83	83	0
first_upstream_to_eci	83	83	0
eci_to_filler	83	83	0
filler_to_pasteurizer	83	83	0
pasteurizer_to_first_downstream	83	83	0
first_to_second_downstream	83	83	0
second_to_third_downstream	83	83	0
third_downstream_to_end	827842	829162	+ 1320
Buffer interno das máquinas	9	9	0
Total	998680	1000000	-
Faltam	**1320**	**0**	**-**

Neste teste foi possível perceber um comportamento transitório na inicialização da linha, que fez com que os *buffers* atingissem o pico em 497 unidades, como apresentado na Figura 4.13. Quando o sistema entra no estado estacionário, os *buffers* estabilizam com 83 unidades, que é o quanto a máquina mais lenta movimenta por ciclo (Equação 4.3). Como nesse teste a primeira máquina é a mais lenta, ela dita a velocidade de toda a linha. A Figura 4.14 apresenta a visão geral dos *buffers* em todo o período do teste.

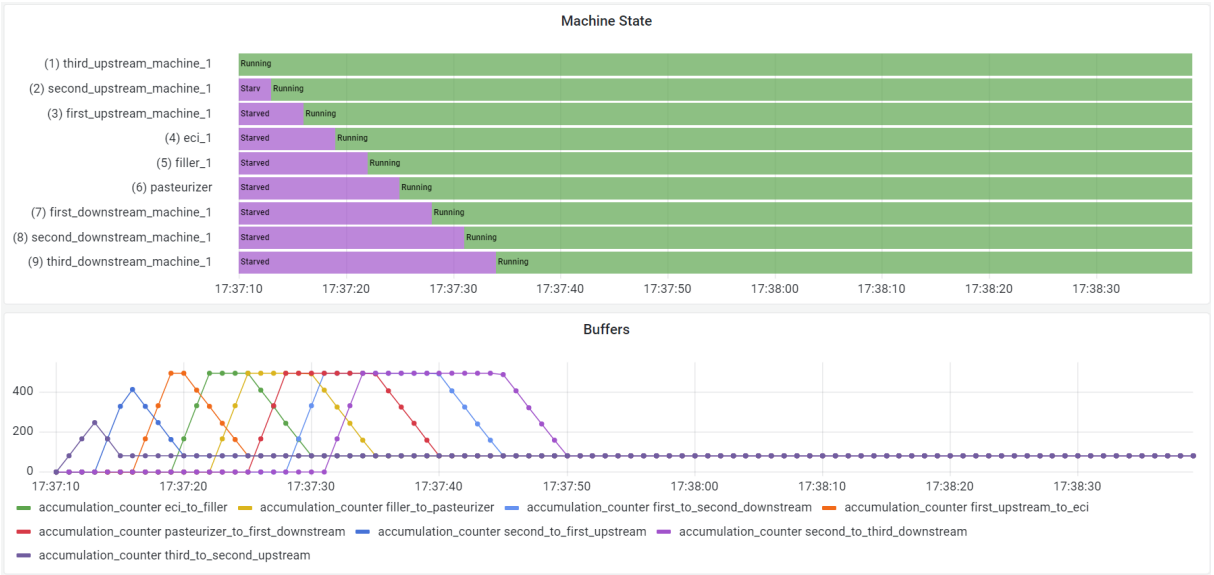

Figura 4.13 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 6.

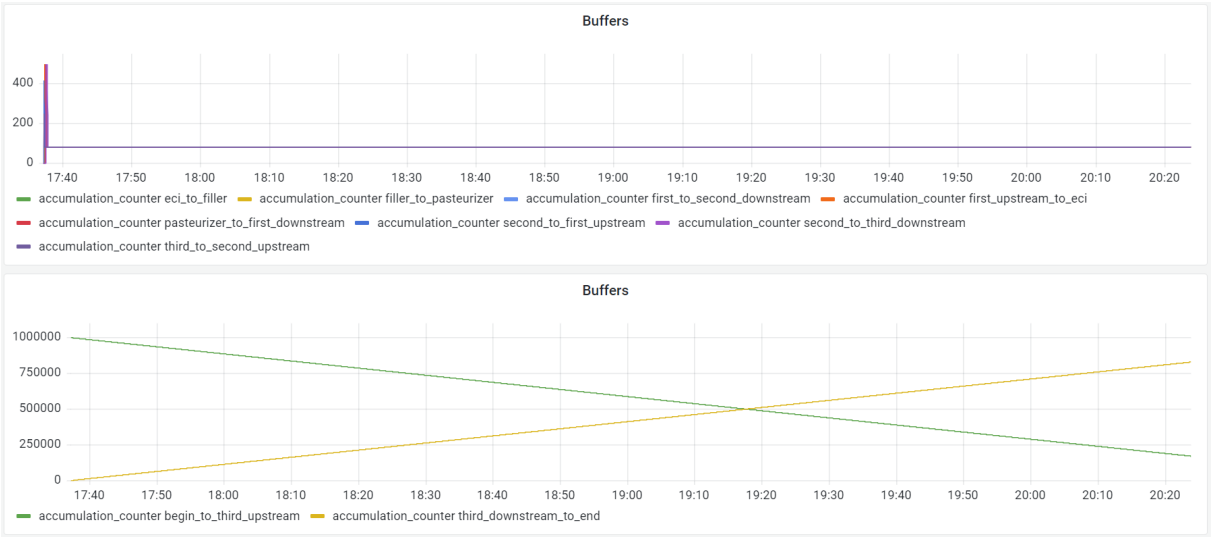

Figura 4.14 – Gráfico com o comportamento dos *buffers* durante todo o Teste 6.

4.1.7 Fase 1: Linhas simples - Teste 7

Este teste, assim como o Teste 6 altera a velocidade apenas da primeira máquina da linha, mas desta vez ela é mais rápida que o restante da linha com velocidade de 20k CPM. A Tabela 4.13 apresenta a configuração das máquinas.

Tabela 4.13 – Configuração das máquinas - Teste 7.

Grupo de máquinas	**Quantidade de máquinas habilitadas**	**Tamanho *buffer* interno**	**Velocidade nominal (CPM)**
Third upstream machine	1	1	20000
Second upstream machine	1	1	10000
First upstream machine	1	1	10000
Empty Container Inspector (ECI)	1	1	10000
Filler	1	1	10000
Pasteurizer	1	1	10000
First downstream machine	1	1	10000
Second downstream machine	1	1	10000
Third downstream machine	1	1	10000

Neste caso, os 10000 ciclos foram suficientes para processar todas as unidades de entrada. Diferente do que foi apresentado nos testes anteriores, os erros encontrados anteriormente não impactaram essa simulação.

Foi possível perceber um comportamento transitório na inicialização da linha no segundo *buffer*, que está localizado imediatamente após a máquina mais rápida. Esse *buffer* teve um pico em 998 unidades, e em seguida estabiliza em 834 unidades. Todos os outros *buffers* estabilizam em 497 unidades, como no Teste 2. A Figura 4.15 apresenta o comportamento descrito anteriormente.

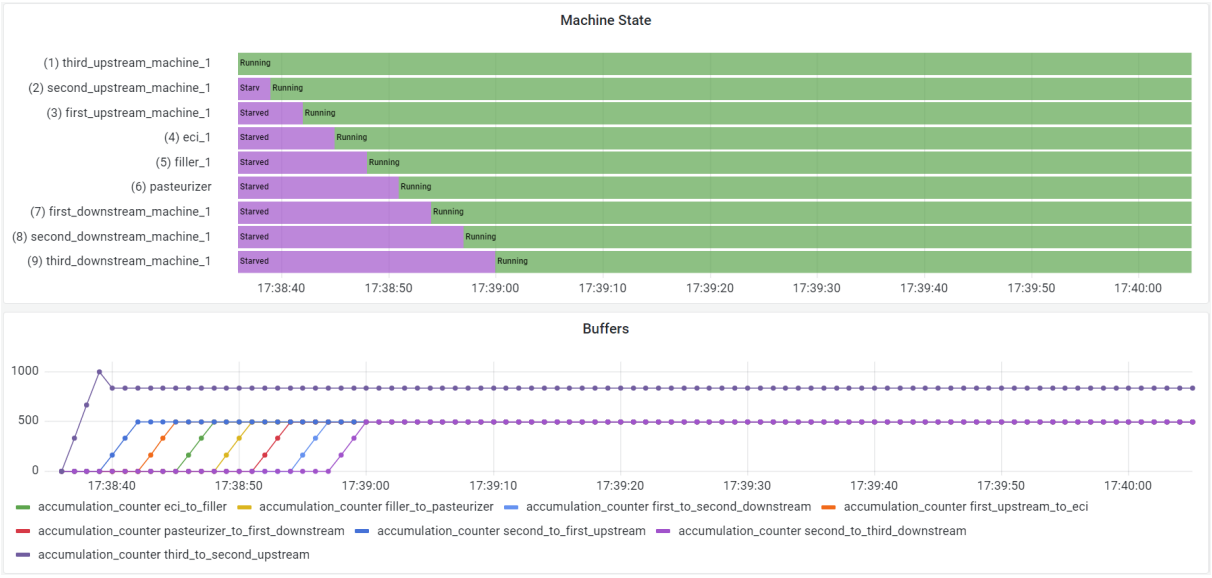

Figura 4.15 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 7.

Outro ponto importante foi observado na finalização do teste. Quando as unidades no *buffer* de entrada terminam, as máquinas mudam o estado de *Running* para *Starved* uma a uma, de acordo com a quantidade de unidades disponíveis no *buffer* anterior. Como o segundo *buffer* tem mais unidades que os demais, é possível perceber que o intervalo entre a mudança de estado da primeira e da segunda máquina é maior que o intervalo entre as demais, como mostra a Figura 4.16. A Figura 4.17 apresenta a visão geral dos *buffers* em todo o período do teste.

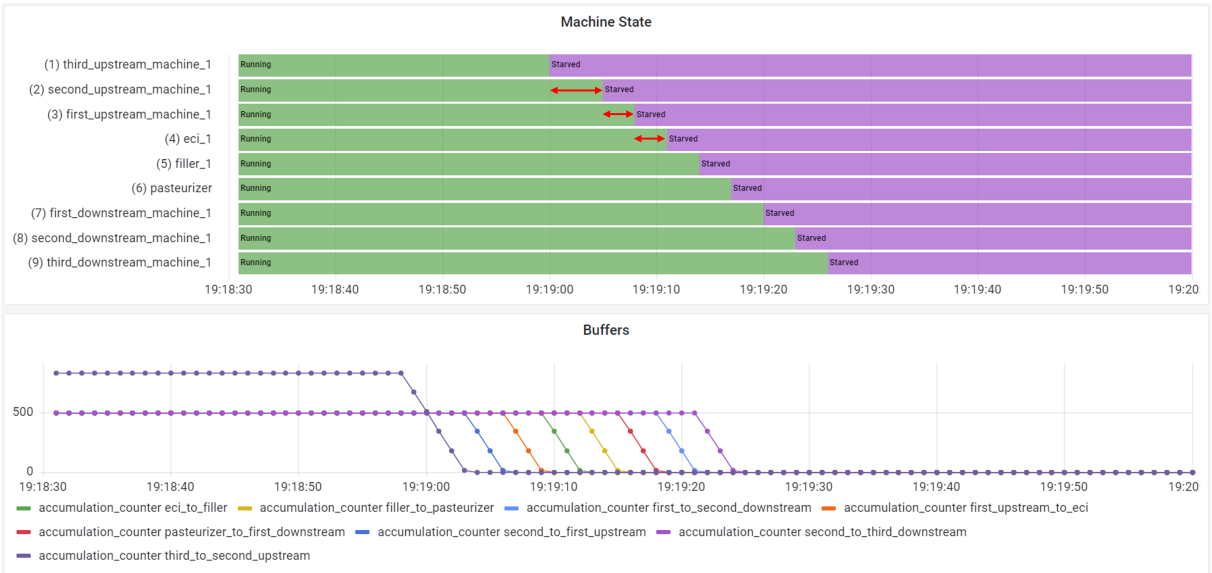

Figura 4.16 – Gráfico do estado das máquinas e do contador dos *buffers* na finalização das unidades no Teste 7.

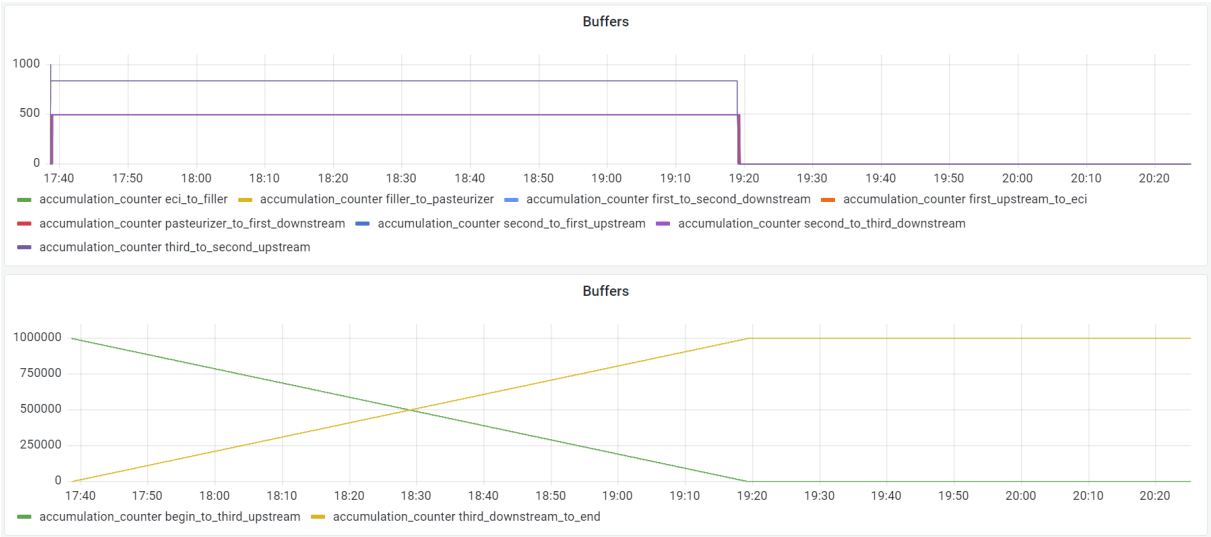

Figura 4.17 – Gráfico com o comportamento dos *buffers* durante todo o Teste 7.

4.1.8 Fase 1: Linhas simples - Teste 8

Conforme apresentado no Teste 2, todas as máquinas foram configuradas com velocidade de 20k CPM e, durante a inicialização da linha, os *buffers* atingiram pico em 998 unidades. Com o intuito de estressar um pouco mais o sistema, neste último teste de validação da linha simples, os *buffers* foram configurados com tamanho máximo de 500 unidades, conforme mostra a Tabela 4.14. A configuração das máquinas foi a mesma do Teste 2, apresentada na Tabela 4.4, e a configuração da simulação permaneceu a mesma, apresentada na Tabela 4.3. Com essa configuração, é esperado que as máquinas entrem no estado *Blocked* em alguns momentos.

Tabela 4.14 – Configuração dos *buffers* - Teste 8.

Nome do *buffer*	**Tamanho máximo**
begin_to_third_upstream	500
third_to_second_upstream	500
second_to_first_upstream	500
first_upstream_to_eci	500
eci_to_filler	500
filler_to_pasteurizer	500
pasteurizer_to_first_downstream	500
first_to_second_downstream	500
second_to_third_downstream	500
third_downstream_to_end	500

Conforme apresentado nos Testes 4, 5 e 6, foram encontrados erros na lógica do simulador base e isso também impactou no comportamento da linha para este cenário.

Durante a análise dos primeiros resultados deste teste, percebeu-se que o limite máximo de 500 unidades dos *buffers* não foi respeitado. A Figura 4.18 apresenta o gráfico dos contadores de cada um dos *buffers*, onde é possível perceber que em determinados momentos alguns deles ultrapassam o valor de 750 e atingem o pico em 831 unidades.

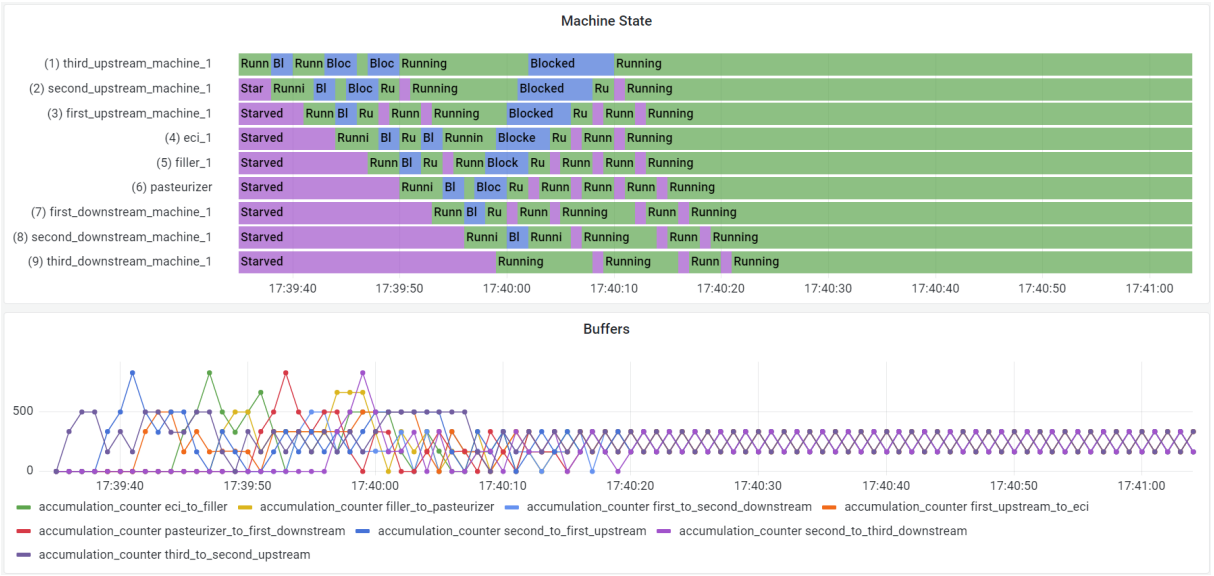

Figura 4.18 – Gráfico do contador dos *buffers* na inicialização da linha na primeira execução do Teste 8.

Após as correções realizadas para solucionar os erros apresentados nos Teste 4 e 5, a simulação deste cenário de teste foi refeita. Os dados da nova simulação foram analisados e constatou-se que a linha apresentou o comportamento esperado. A Figura 4.19 apresenta o gráfico do estado das máquinas e o contador dos *buffers* na inicialização do teste. É possível perceber que, assim como já estava previsto, as máquinas assumem o estado *Blocked* em alguns momentos e os *buffers* não ultrapassam o valor máximo definido para o teste.

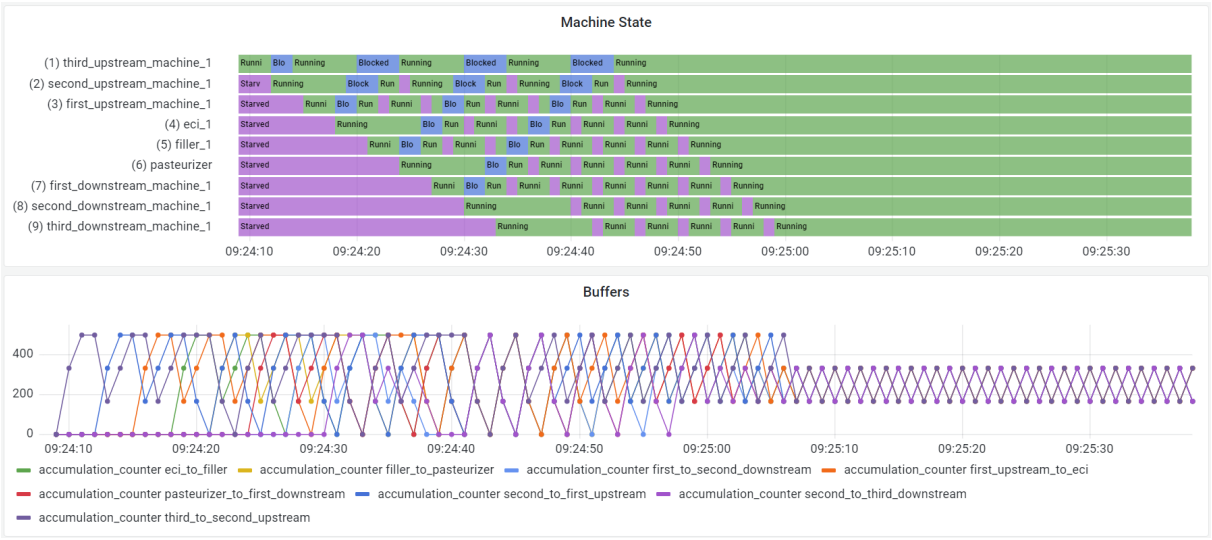

Figura 4.19 – Gráfico do estado das máquinas e do contador dos *buffers* na finalização das unidades no Teste 8.

A Figura 4.20 apresenta a visão geral dos *buffers* em todo o período do teste, onde é possível perceber que os 10000 ciclos foram suficientes para processar todas as unidades de entrada. Como os *buffers* não ficam com um valor constante no regime estacionário, os contadores oscilam entre 167 e 333 unidades a cada ciclo. A oscilação entre os valores faz com que o primeiro gráfico tenha um formato diferente em relação aos testes anteriores e apresente uma linha mais grossa. A Figura 4.19 apresenta os mesmos dados com um zoom maior no gráfico, onde é possível verificar a oscilação dos valores.

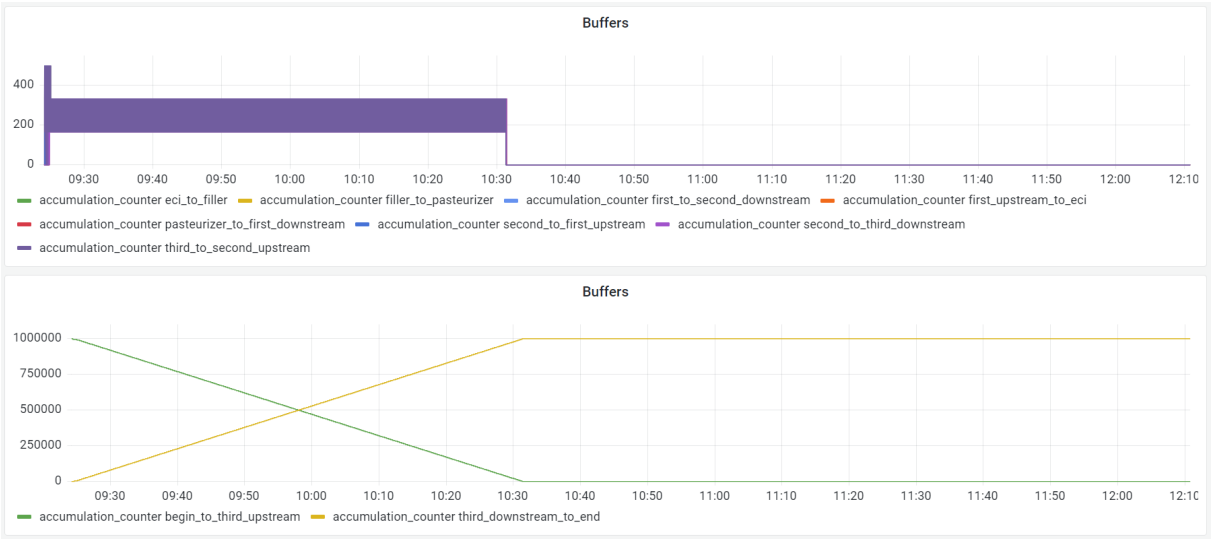

Figura 4.20 – Gráfico com o comportamento dos *buffers* durante todo o Teste 8.

4.2 Fase 2: Linhas com Ramificações

A segunda fase de validação do PLSim tem como objetivo testar a ramificação das linhas, que é quando há mais de uma máquina habilitada em pelo menos um grupo. Nessa configuração, as máquinas do mesmo grupo operam em paralelo e os *buffers* de entrada e saída são os mesmos.

Para a definição dos cenários de teste, a estratégia usada consistiu em escolher configurações que se assemelhassem as configurações dos testes realizados na primeira etapa, para que o comportamento do PLSim pudesse ser comparado e validado. Para exemplificar, uma máquina com velocidade de 20k CPM na primeira etapa, foi substituída por duas máquinas de velocidade 10k CPM na segunda fase dos testes. Assim a velocidade do grupo permanece a mesma e os resultados obtidos devem ser semelhantes. Assim como na primeira fase de teste, os valores dos parâmetros MTBF e MTTR foram definidos como zero em todos os cenários de teste.

Foram definidos quatro cenários de teste e, diferente da abordagem utilizada na primeira fase, cada teste foi executado e analisado, assim, os erros encontrados eram corrigidos antes da execução do próximo cenário de teste. Foram encontrados alguns erros durante as simulações que serão apresentados nas seções a seguir.

A Figura 4.21 apresenta os cenários de teste propostos nesta fase. O Teste 9 adiciona outra máquina gargalo, identificada como *Filler* na figura. O Teste 10 adiciona outra máquina no primeiro grupo, identificado como *Third Upstream*. O Teste 11 adiciona a segunda máquina em todos os grupos, exceto no grupo identificado como *Pasteurizer*, que só permite uma por linha. Para manter todos os grupos com velocidade total de 20k CPM, a velocidade do *Pasteurizer* foi alterada para 20k CPM. O último cenário, Teste 12, adiciona outras três máquinas no primeiro grupo e configura todas as máquinas deste grupo com velocidade de 5k CPM.

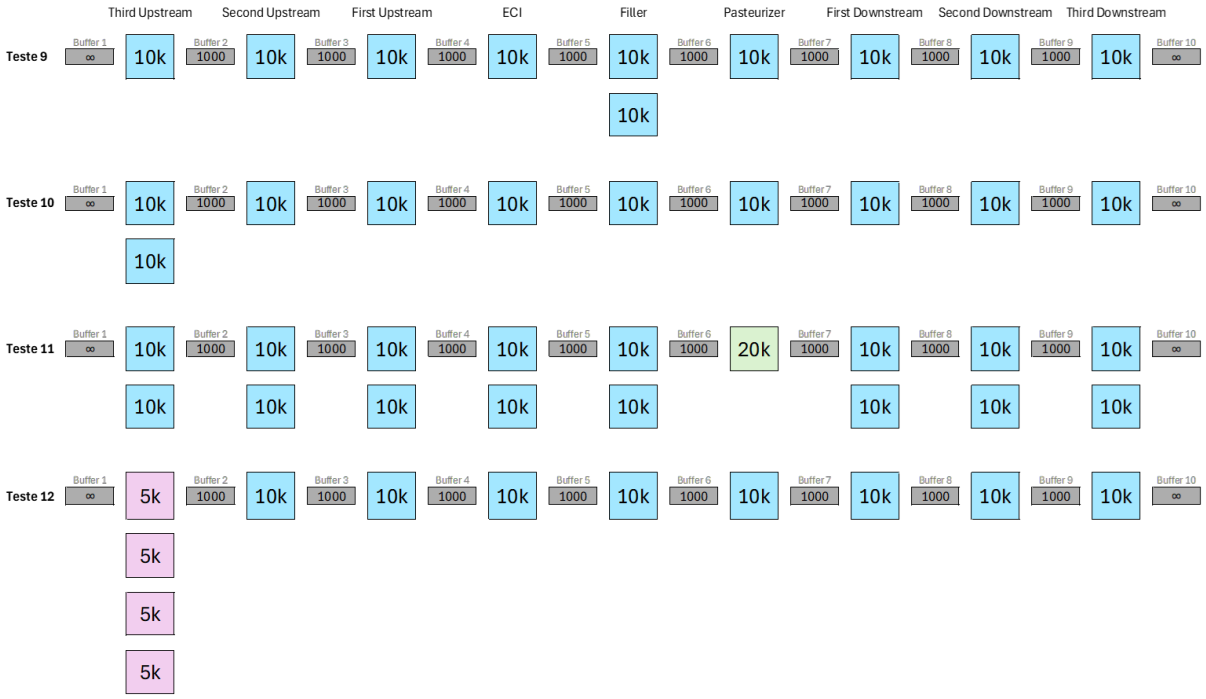

Figura 4.21 – Representação da linha de envase configurada nos Testes 9 a 12.

4.2.1 Fase 2: Linhas com Ramificações - Teste 9

Este teste tinha como referência o Teste 5, em que todas as máquinas foram configuradas com velocidade de 10k CPM e o *Filler* com velocidade de 20k CPM. Neste caso, a linha contém dois *fillers* com velocidade de 10k CPM cada, conforme apresentado na Tabela 4.15. A configuração dos *buffers* e da simulação permaneceram iguais as do Teste 1, apresentadas nas Tabelas 4.2 e 4.3.

Tabela 4.15 – Configuração das máquinas - Teste 9.

Grupo de máquinas	Quantidade de máquinas habilitadas	Tamanho *buffer* interno	Velocidade nominal (CPM)
Third upstream machine	1	1	10000
Second upstream machine	1	1	10000
First upstream machine	1	1	10000
Empty Container Inspector (ECI)	1	1	10000
Filler	2	1	10000
Pasteurizer	1	1	10000
First downstream machine	1	1	10000
Second downstream machine	1	1	10000
Third downstream machine	1	1	10000

O resultado da primeira execução do teste não foi o esperado. Houveram algumas divergências nos valores dos *buffers* quando comparados com os resultados do Teste 5 e, na tentativa de entender o motivo das diferenças, percebeu-se que as duas máquinas do mesmo grupo não iniciavam juntas. Após uma investigação foram encontrados dois erros no simulador, que são apresentados a seguir.

O primeiro erro estava na ordem que a lógica da simulação era executada. Os objetos que representam as máquinas, classe *Machine*, são instanciados em uma lista, e cada um deles possui referência para os objetos dos *buffers* de entrada e de saída. Os objetos que representam os *buffers*, classe *AccumulationBuffer*, também são instanciados em uma lista, e cada um deles possui duas listas internas, uma com as máquinas de entrada e outra com as máquinas de saída do *buffer*. Inicialmente, a simulação executava um *loop for* na lista de máquinas e, em cada iteração, os métodos com a lógica do objeto máquina eram chamados seguidos dos métodos do objeto salvo como *buffer* de entrada. Na situação em que tem-se mais de uma máquina no mesmo grupo, a lógica do *buffer* de entrada estava sendo executada mais de uma vez no mesmo ciclo. Para corrigir esse problema, o *loop for* passou a ser executado na lista de *buffers* e, para cada iteração, faz-se um *loop for* na lista de máquinas de saída, onde os métodos do objeto são chamados, e, em seguida, são chamados os métodos do *buffer*.

O segundo problema estava relacionado com o fato das máquinas de um mesmo grupo não inicializarem ao mesmo tempo, mesmo quando há unidades suficientes para todas elas no *buffer* de entrada. No início do método que executa a lógica do comportamento das máquinas era chamado um método que retorna a quantidade de unidades disponíveis no *buffer* de entrada para aquela máquina. Esse valor varia de acordo com a quantidade e velocidade do grupo de máquinas que tem aquele *buffer* como *buffer* de entrada. Como a execução da lógica das máquinas é sequencial, em alguns momentos, o valor retornado pelo método era diferente. Para que todas as máquinas de um mesmo grupo sempre utilizassem o mesmo valor no ciclo, esse valor passou a ser calculado antes da execução da lógica das máquinas daquela iteração.

Após essas duas alterações o teste foi refeito e, desta vez, os resultados foram os esperados. A Figura 4.22 apresenta o comportamento das máquinas e dos *buffers* na inicialização do teste, que é semelhante ao resultado do Teste 5. A única diferença entre eles é o valor que o *buffer filler_to_pasteurizer* estabiliza no regime estacionário. No Teste 5 o *buffer* estabilizou em 828 unidades e neste teste foi em 827. Essa diferença de uma unidade é devido ao *buffer* interno da segunda máquina adicionada. A Figura 4.23 apresenta a visão geral dos *buffers* em todo o período do teste.

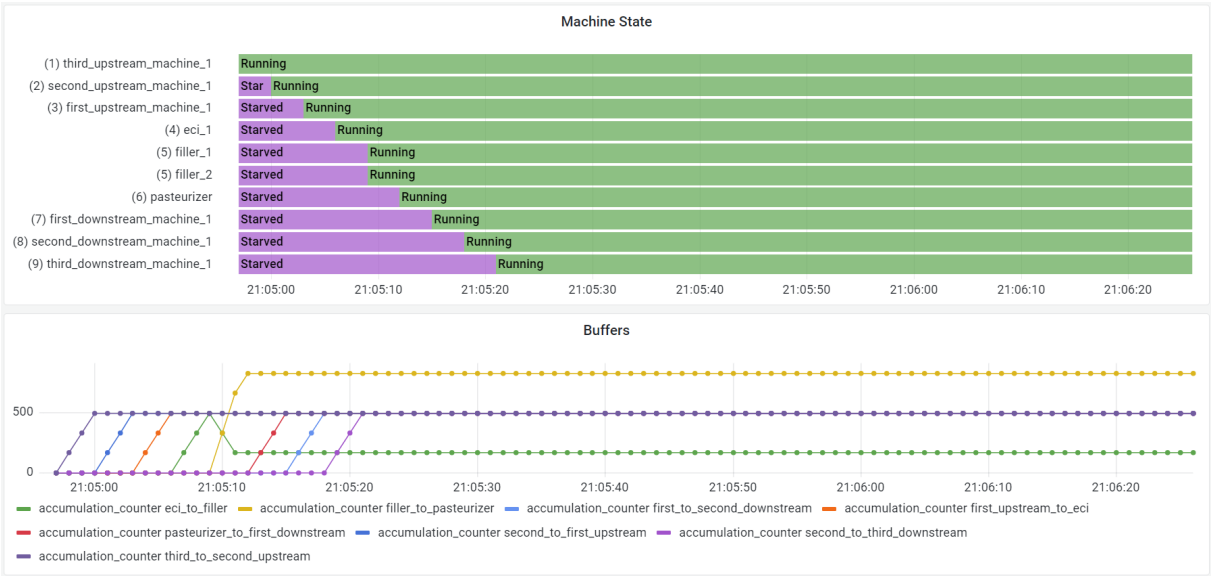

Figura 4.22 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 9.

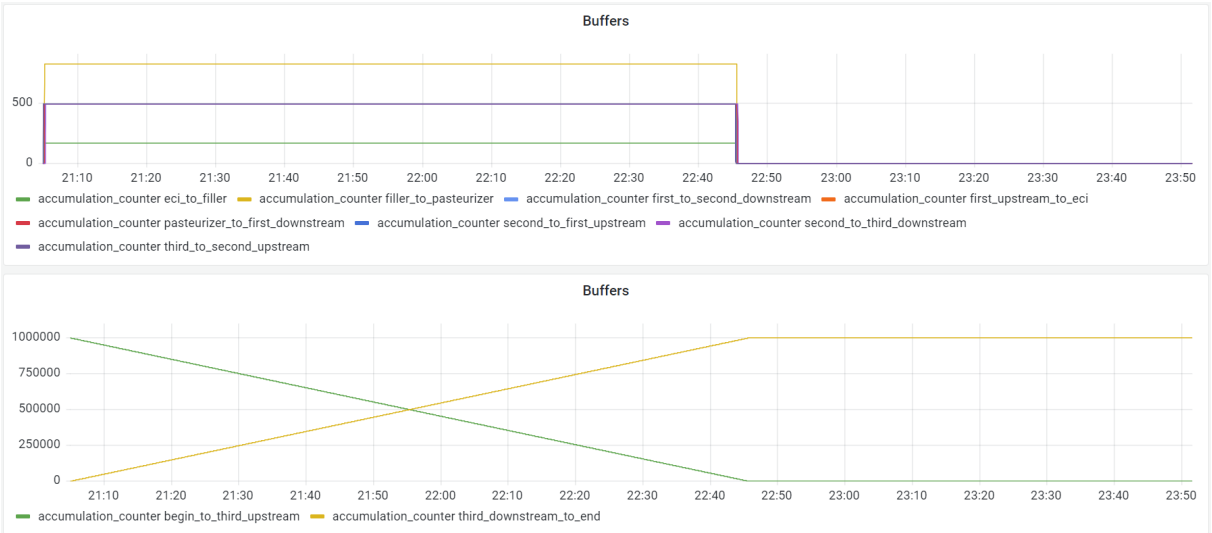

Figura 4.23 – Gráfico com o comportamento dos *buffers* durante todo o Teste 9.

4.2.2 Fase 2: Linhas com Ramificações - Teste 10

Este teste tem como referência o Teste 7, em que todas as máquinas são configuradas com velocidade de 10k CPM, exceto a primeira, que foi configurada com velocidade de 20k CPM. Neste caso, a linha contém duas *Third Upstream Machine* com velocidade de 10k CPM cada, conforme apresentado na Tabela 4.16. A configuração dos *buffers* e da simulação permaneceram iguais as do Teste 1, apresentadas nas Tabelas 4.2 e 4.3.

Tabela 4.16 – Configuração das máquinas - Teste 10.

Grupo de máquinas	Quantidade de máquinas habilitadas	Tamanho *buffer* interno	Velocidade nominal (CPM)
Third upstream machine	2	1	10000
Second upstream machine	1	1	10000
First upstream machine	1	1	10000
Empty Container Inspector (ECI)	1	1	10000
Filler	1	1	10000
Pasteurizer	1	1	10000
First downstream machine	1	1	10000
Second downstream machine	1	1	10000
Third downstream machine	1	1	10000

Quando comparado o resultado obtido neste teste com o Teste 7, percebeu-se uma diferença no gráfico que apresenta o comportamento dos *buffers* na simulação, o valor em que os *buffers* estabilizavam estavam diferentes. Após uma análise detalhada da simulação, a causa foi identificada. O problema era bem parecido com o primeiro erro descrito no Teste 9, mas desta vez para a saída da máquina. Na lógica da máquina, uma vez que ela já está inicializada e seu *buffer* interno cheio, ela verifica quantas unidades cabem no *buffer* de saída para decidir quantas unidades serão processadas naquele ciclo. Na implementação original, cada máquina entendia que poderia usar o valor total. Para exemplificar, supondo que o *buffer* de saída só tem espaço para dez unidades naquele momento, o correto é que o grupo de máquinas localizado antes do *buffer* divida esse valor entre as máquinas, e não que cada máquina deposite dez unidades, pois assim o limite máximo do *buffer* será ultrapassado.

Para solucionar o comportamento errado descrito no parágrafo anterior, haviam duas estratégias. A primeira seria dar prioridade para as máquina na ordem de execução, assim, a primeira tem maior prioridade em relação a segunda e assim sucessivamente. A segunda estratégia seria dividir o espaço disponível entre as máquinas, portanto, no exemplo, cada máquina usaria cinco unidades. As duas estratégias foram implementadas e testada.

Neste teste chega o momento em que um *buffer* fica com apenas cinco unidades disponíveis e por isso essa questão foi identificada. Na primeira estratégia, quando ocorre essa situação, a segunda máquina vai para o estado *Blocked* e fica assim até o fim da simulação. Já na segunda estratégia, cada máquina utiliza duas unidades do *buffer* e ambas permanecem em operação durante toda a simulação. Para este trabalho, após consultar a opinião de um especialista em linhas de envase, optou-se por deixar a segunda estratégia implementada para a apresentação dos resultados e execução dos demais testes. Após essa correção, o Teste 9 foi refeito, mas essa alteração não impactou a simulação e os resultados continuaram os mesmos.

Após essa alteração o teste foi refeito e, desta vez, os resultados foram os esperados. A Figura 4.24 apresenta o comportamento das máquinas e dos *buffers* na inicialização do teste, que é semelhante ao resultado do Teste 7. A única diferença entre eles é o valor

que o *buffer third_to_second_upstream* estabiliza no regime estacionário. No Teste 7 o *buffer* estabilizou em 834 unidades e neste teste foi em 833. Essa diferença de uma unidade é devido ao *buffer* interno da segunda máquina adicionada. A Figura 4.25 apresenta a visão geral dos *buffers* em todo o período do teste.

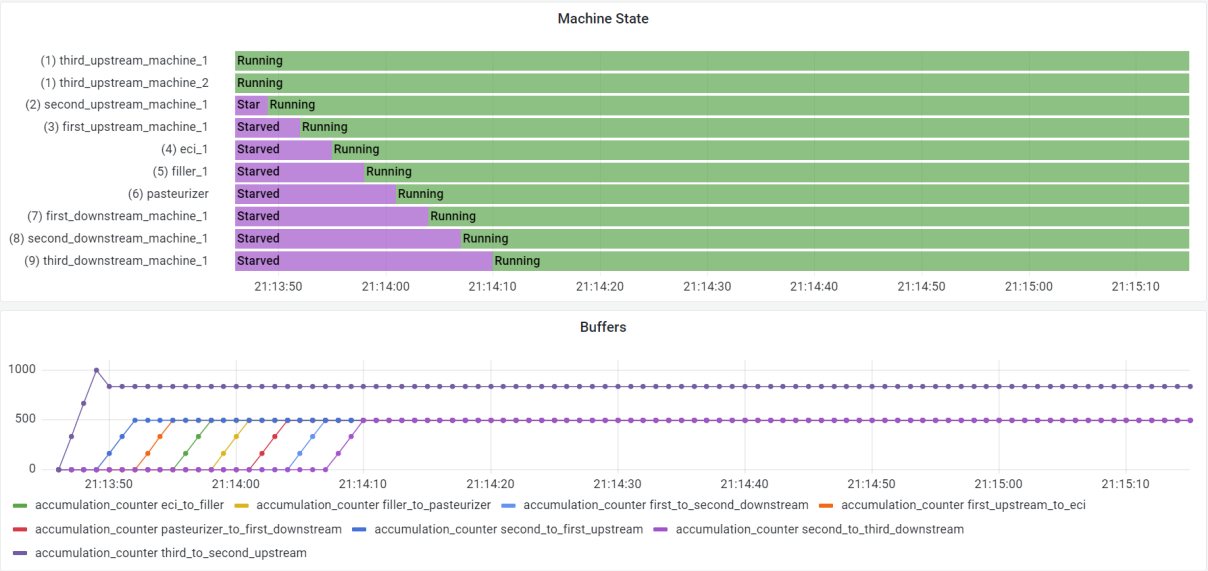

Figura 4.24 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 10.

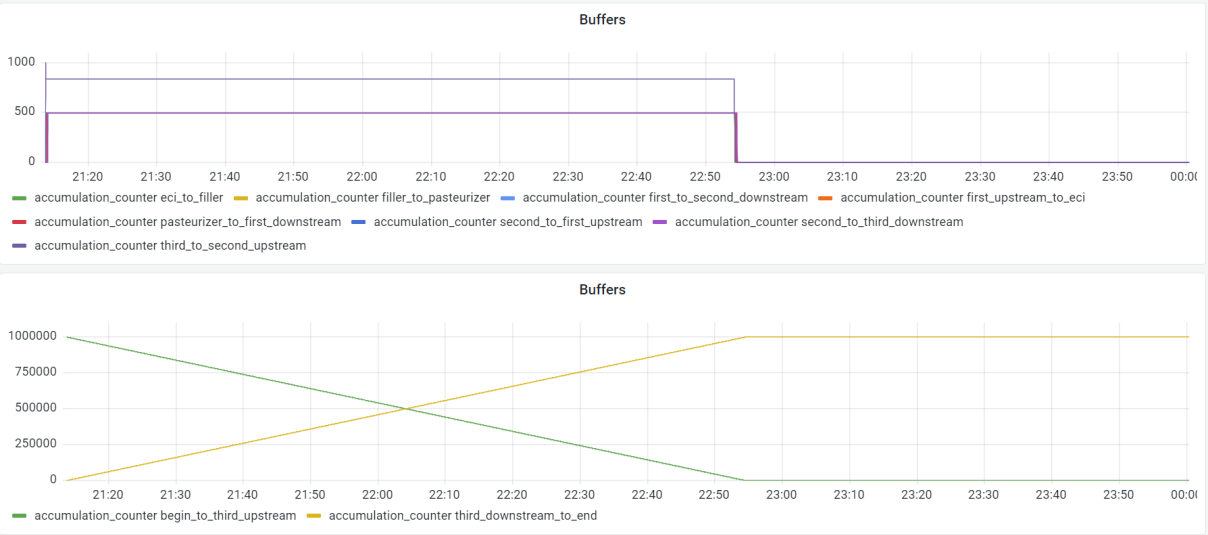

Figura 4.25 – Gráfico com o comportamento dos *buffers* durante todo o Teste 10.

4.2.3 Fase 2: Linhas com Ramificações - Teste 11

Este teste tem como referência o Teste 2, em que todas as máquinas são configuradas com velocidade de 20k CPM. Neste caso, a linha contém duas máquinas em cada grupo com velocidade de 10k CPM, exceto pelo *Pasteurizer*, que só é permitido ter um por linha, que a velocidade foi configurada com 20k CPM, conforme apresentado na Tabela 4.17. A configuração dos *buffers* e da simulação permaneceram iguais as do Teste 1, apresentadas nas Tabelas 4.2 e 4.3.

Tabela 4.17 – Configuração das máquinas - Teste 11.

Grupo de máquinas	**Quantidade de máquinas habilitadas**	**Tamanho *buffer* interno**	**Velocidade nominal (CPM)**
Third upstream machine	2	1	10000
Second upstream machine	2	1	10000
First upstream machine	2	1	10000
Empty Container Inspector (ECI)	2	1	10000
Filler	2	1	10000
Pasteurizer	1	1	20000
First downstream machine	2	1	10000
Second downstream machine	2	1	10000
Third downstream machine	2	1	10000

Os resultados obtidos neste teste foram os esperados. A Figura 4.26 apresenta o comportamento das máquinas e dos *buffers* na inicialização do teste, que é semelhante ao resultado do Teste 2. O formato do gráfico dos *buffers* é igual, porém, os valores estão um pouco diferentes. A primeira diferença é que no Teste 2 cada máquina processa 333 CPS, já neste teste cada máquina processa 166 CPS e, somando o valor das duas máquinas, o grupo processa 332 CPS. E a segunda diferença está relacionada com *buffer* interno das máquinas adicionadas. Neste teste, durante o período transitório no início da simulação, o número de unidades em cada *buffer* variou entre 664 e 994. Já no regime estacionário, todos os *buffers* estabilizaram com 668 unidades, exceto o *buffer filler_to_pasteurizer* que ficou com 667. A Figura 4.27 apresenta a visão geral dos *buffers* em todo o período do teste.

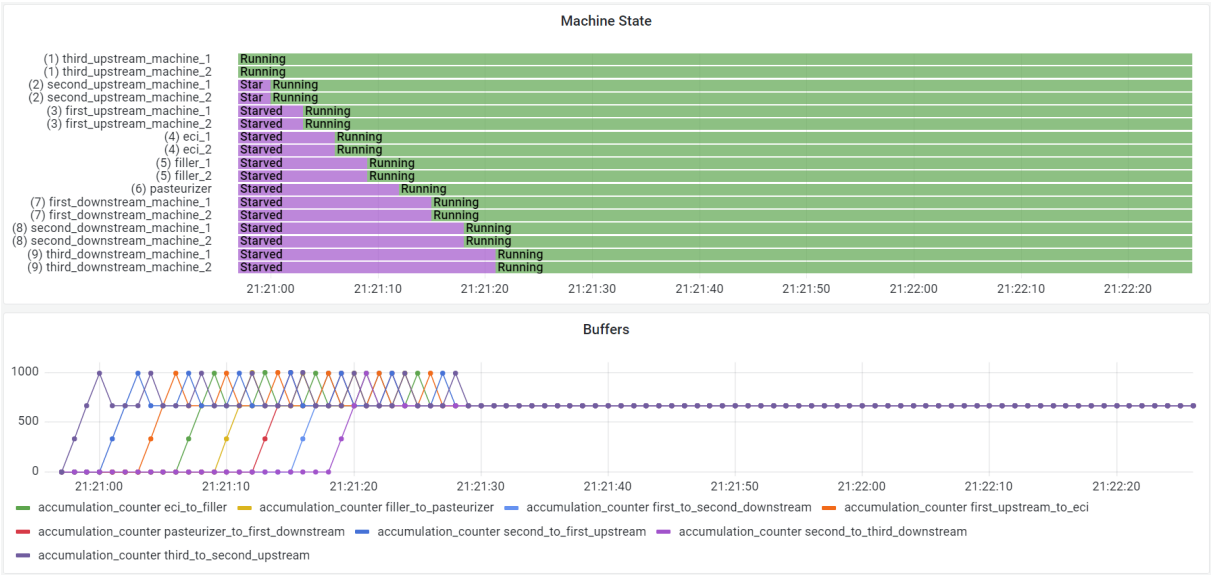

Figura 4.26 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 11.

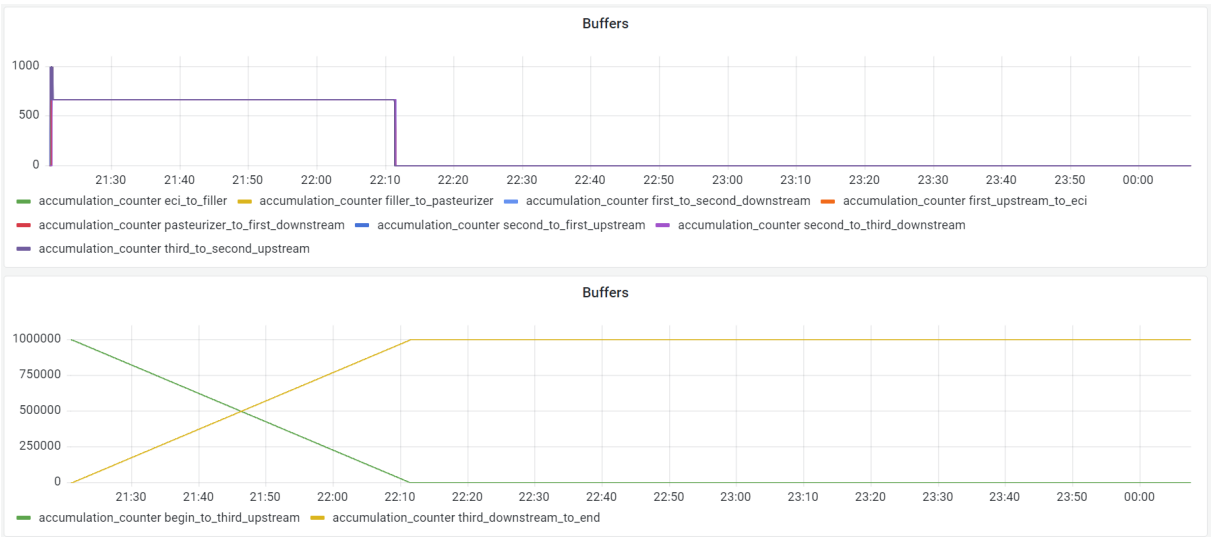

Figura 4.27 – Gráfico com o comportamento dos *buffers* durante todo o Teste 11.

4.2.4 Fase 2: Linhas com Ramificações - Teste 12

Este teste, assim como o Teste 10, tinha como referência o Teste 7, em que todas as máquinas foram configuradas com velocidade de 10k CPM, exceto a primeira, que foi configurada com velocidade de 20k CPM. Neste caso, a linha contém quatro *Third Upstream Machine* com velocidade de 5k CPM cada, conforme apresentado na Tabela 4.18. O objetivo deste teste era validar a simulação com mais de duas máquinas em um

mesmo grupo. A configuração dos *buffers* e da simulação permaneceram iguais as do Teste 1, apresentadas nas Tabelas 4.2 e 4.3.

Tabela 4.18 – Configuração das máquinas - Teste 12.

Grupo de máquinas	Quantidade de máquinas habilitadas	Tamanho *buffer* interno	Velocidade nominal (CPM)
Third upstream machine	4	1	5000
Second upstream machine	1	1	10000
First upstream machine	1	1	10000
Empty Container Inspector (ECI)	1	1	10000
Filler	1	1	10000
Pasteurizer	1	1	10000
First downstream machine	1	1	10000
Second downstream machine	1	1	10000
Third downstream machine	1	1	10000

Os resultados obtidos neste teste foram os esperados. A Figura 4.28 apresenta o comportamento das máquinas e dos *buffers* na inicialização do teste, que é semelhante ao resultado dos Testes 7 e 10. A única diferença entre eles é o valor que o *buffer* *third_to_second_upstream* estabiliza no regime estacionário. No Teste 7, o *buffer* estabilizou em 834 unidades e neste teste foi 831. Essa diferença de três unidades é devido ao *buffer* interno das três máquinas adicionadas. A Figura 4.29 apresenta a visão geral dos *buffers* em todo o período do teste.

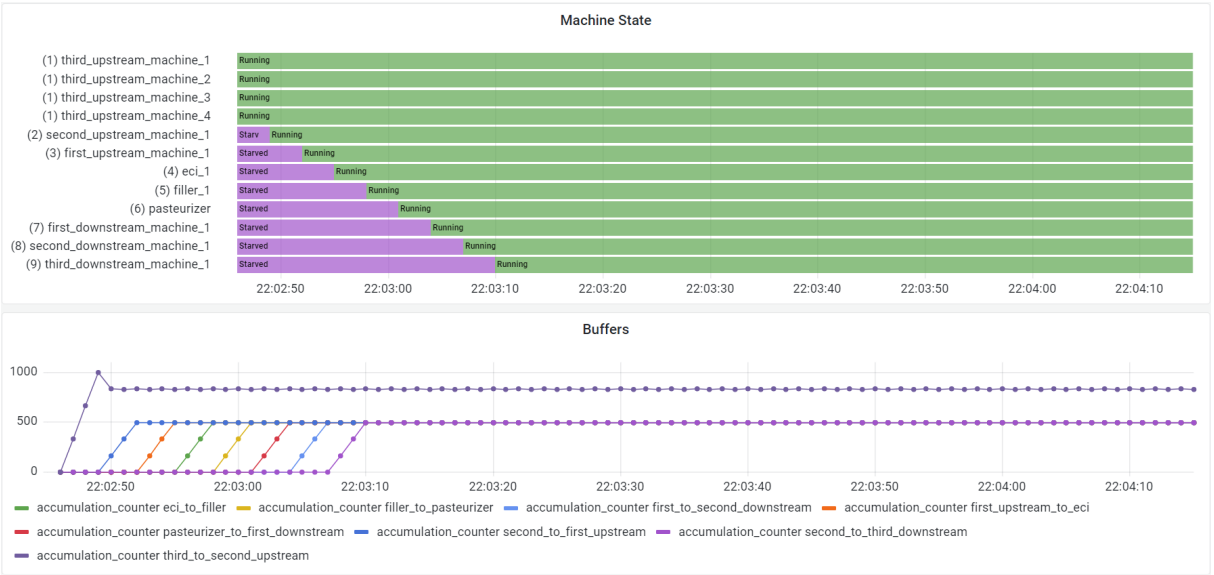

Figura 4.28 – Gráfico do estado das máquinas e do contador dos *buffers* na inicialização da linha no Teste 12.

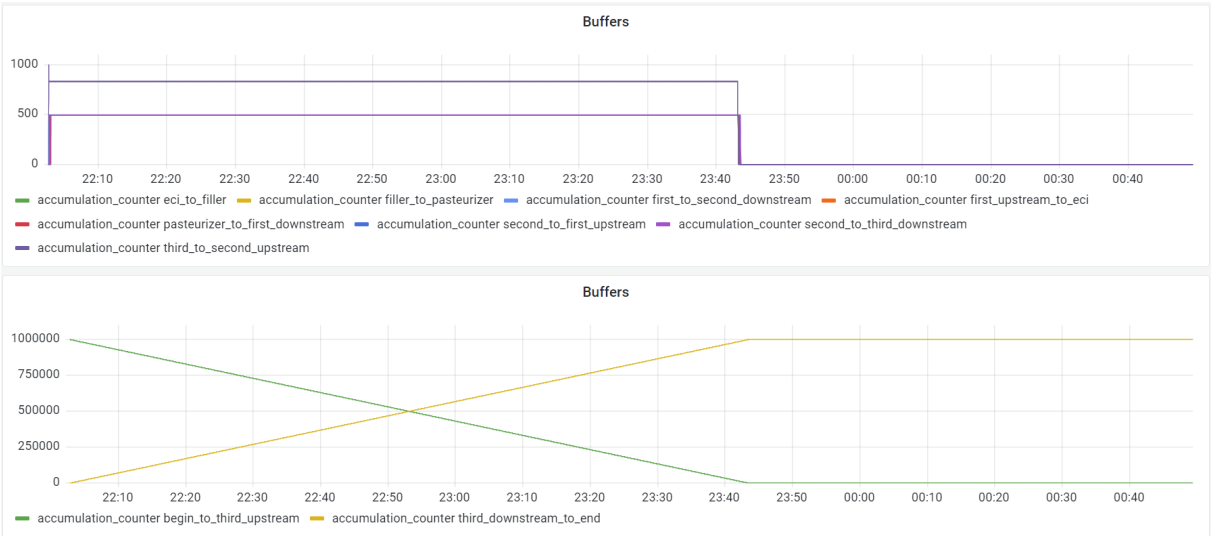

Figura 4.29 – Gráfico com o comportamento dos *buffers* durante todo o Teste 12.

4.3 Fase 3: Falhas Aleatórias

A terceira, e última, fase de testes tinha como objetivo testar as falhas aleatórias no PLSim. Conforme apresentado na Seção 3.1, foi implementado um método no código do simulador que define o início e o fim das falhas aleatórias de acordo com os valores definidos nos parâmetros MTBF e MTTR de cada máquina. Essa etapa, além de testar se o método proposto estava funcionando de acordo com o esperado, verificou como o restante dos objetos das linha se comportavam.

Os valores de MTBF e MTTR utilizados nessa fase de teste foram obtidos a partir de dados reais de uma linha da cervejaria, em que se considerou dois cenários, gerando dois conjuntos de dados para cada um dos parâmetros. A Seção 4.3.1 apresenta como os dados foram trabalhados para obtenção dos valores de MTBF e MTTR para cada uma das máquinas. Nesta fase de teste foram propostos quatro cenários de teste, dois para cada conjunto de valores de MTBF e MTTR.

4.3.1 Definição dos Parâmetros de Falhas e Microparadas

Os valores dos parâmetros MTBF e MTTR usados nos testes foram obtidos por meio de uma planilha que contém dados de uma linha de envase da cervejaria, no período de um mês. A planilha contém, numa linha do tempo, todos os estados em que cada máquina esteve durante aquele período e quanto tempo ficou em cada estado. Como o cálculo dos parâmetros é feito utilizando o tempo total no numerador, as informações são suficientes para obtenção dos valores.

As opções de estado de máquina na planilha são as mesmas utilizadas nesta dissertação: operação, falta de contêiner, bloqueado e falha. Conforme apresentado na Seção 2.4.2, o cálculo do MTBF utiliza o tempo total da máquina em operação e o MTTR utiliza o tempo total da máquina parada. Como esses são parâmetros de performance da linha e são calculados com dados históricos, é considerado parada os estados 'falta de contêiner', 'bloqueio' e 'falha', que foi o primeiro cenário a ser analisado. Porém, o intuito do PLSim é emular apenas as falhas, pois os estados de falta de contêiner e bloqueio são consequências da dinâmica da linha. A partir dessa análise surgiu o segundo cenário para o cálculo dos parâmetros, que considera como parada apenas os momentos em que máquina está em falha. A Tabela 4.19 resume os cenários descritos, indicando os estados considerados em cada um dos parâmetros.

Tabela 4.19 – Cenários para o cálculo dos parâmetros MTBF e MTTR.

	MTBF	**MTTR**
Cenário 1	Operação	Falta de contêiner, bloqueio e Falha
Cenário 2	Operação, falta de contêiner e bloqueio	Falha

A Tabela 4.20 apresenta os valores dos parâmetros MTBF e MTTR para cada um dos cenários. A linha em questão tem duas enchedoras e duas *first_downstream_machine*. Os valores dos parâmetros apresentados na tabela estão em segundos. É importante destacar que o valor do MTBF ficou bem diferente entre os cenários, como já era esperado, já que ele verifica o tempo médio entre as falhas. Com isso, é possível perceber que as simulações que utilizam os dados do Cenário 2 têm menos paradas que as do Cenário 1.

Tabela 4.20 – Valores dos parâmetros MTBF e MTTR para os dois cenários.

	Cenário 1		**Cenário 2**	
Máquinas	**MTBF**	**MTTR**	**MTBF**	**MTTR**
third_upstream_machine_1	79	38	2063	71
second_upstream_machine_1	260	47	784	46
first_upstream_machine_1	158	20	291	16
eci_1	523	20	567	14
filler_1	425	79	14054	62
filler_2	641	66	1636	63
pasteurizer	157	33	2330	34
first_downstream_machine_1	600	81	3348	136
first_dowstream_machine_2	451	98	3759	118
second_downstream_machine_1	49	15	85	7
third_downstream_machine_1	583	75	711	78

A Figura 4.30 apresenta as duas configurações utilizadas na Fase 3. A primeira, utilizada nos Testes 13 e 15, é uma linha simples com todas as máquinas com velocidade de 10k CPM. Na segunda configuração, utilizada nos Testes 14 e 16, foram adicionadas

duas máquinas, uma no grupo *Filler* e outra no *First Downstream*, para se assemelhar à linha de onde foram obtidos os dados para o cálculo dos parâmetros MTBF e MTTR. Nesses grupos, cada máquina tinha velocidade de 5k CPM. Os Testes 13 e 14 utilizaram os dados do Cenário 1 apresentados na Tabela 4.20, e os Testes 15 e 16 utilizaram os dados do Cenário 2.

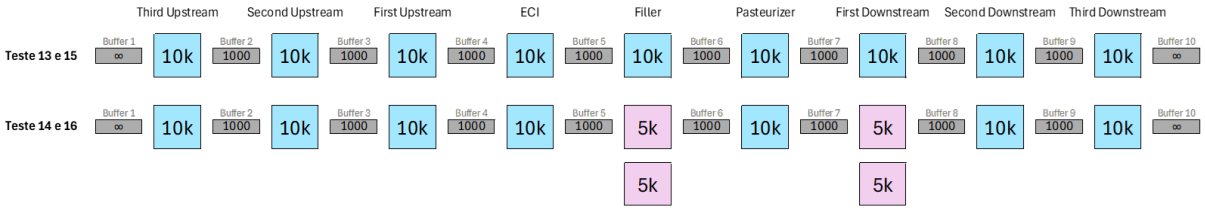

Figura 4.30 – Representação da linha de envase configurada nos Testes 13 a 16.

4.3.2 Fase 3: Falhas Aleatórias - Teste 13

Neste teste, as configurações das máquinas, dos *buffers* e da simulação são iguais as do Teste 1, apresentadas nas Tabelas 4.1, 4.2 e 4.3, respectivamente. A alteração foi feita nos valores do MTBF e MTTR, que no Teste 1 ambos estavam zerados e neste teste foram utilizados os valores do Cenário 1 apresentados na Tabela 4.20.

A Figura 4.31 apresenta o estado das máquinas e dos *buffers* nos primeiros 30 minutos de simulação. Diferente dos gráficos apresentados anteriormente, é possível perceber que, em alguns momentos, as máquinas vão para o estado *Stopped*, representado pela cor vermelha, que são as falhas induzidas no sistema.

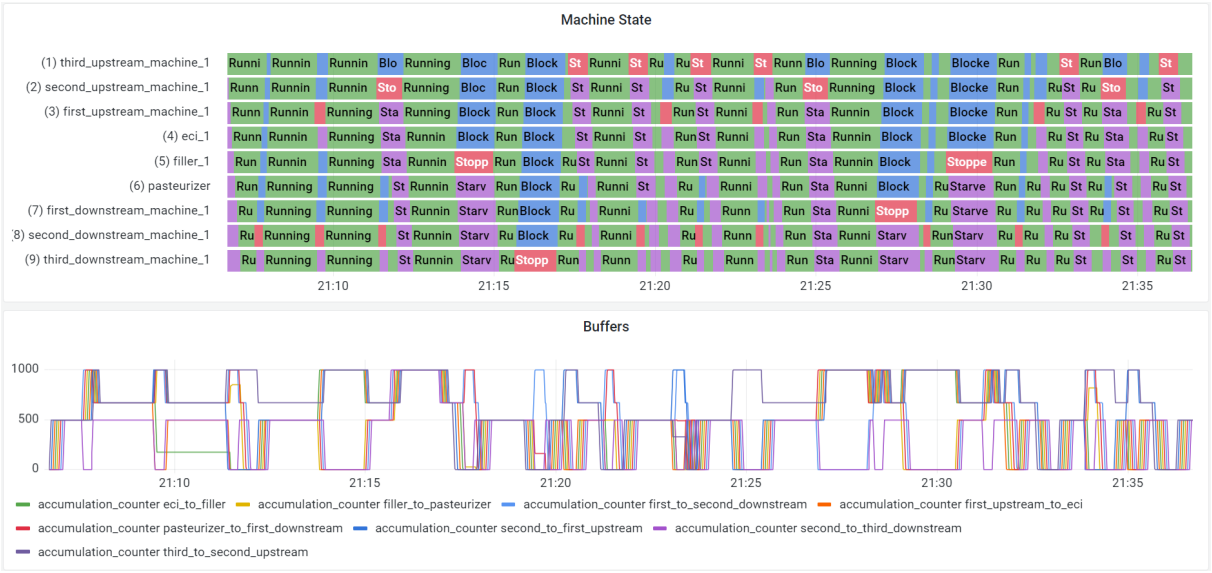

Figura 4.31 – Gráfico do estado das máquinas e do contador dos *buffers* nos primeiros 30 minutos do Teste 13.

De modo geral, quando uma falha acontece, as máquinas que estão antes da máquina parada vão para o estado *Blocked*, porque há um acúmulo de contêineres nos *buffers*, e as máquinas que estão depois vão para o estado *Starved*, porque os *buffers* estão vazios. O gráfico dos *buffers* da Figura 4.31 ajuda a entender essa dinâmica, onde é possível perceber que, em vários momentos os *buffers* atingem seus valores máximos e mínimos, 1000 e 0, respectivamente. Com o intuito de detalhar a dinâmica descrita, foi selecionado um período de pouco mais de 3 minutos, onde acontecem três falhas, conforme apresentado na Figura 4.32. Os momentos marcados no gráfico estão descritos a seguir.

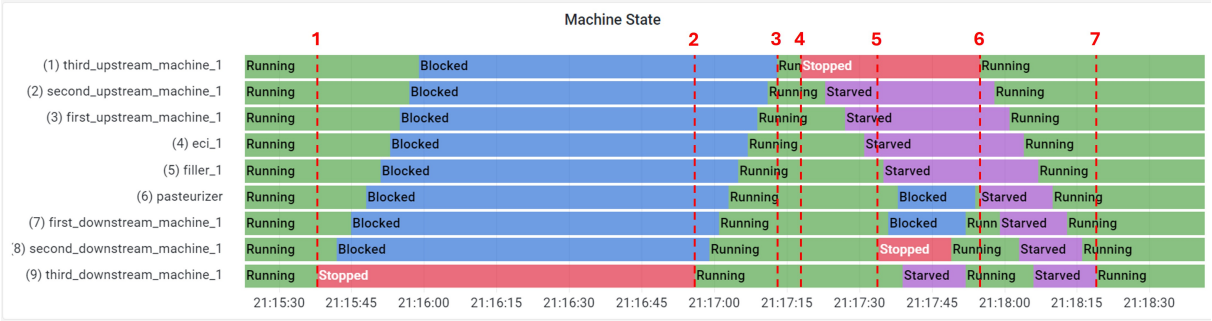

Figura 4.32 – Gráfico com o comportamento das máquinas durante falhas no Teste 13.

- Em 1, a primeira falha é iniciada na última máquina e, em seguida, as máquinas anteriores vão mudando para o estado *Blocked* uma por uma. Com a última máquina parada, o *buffer* de entrada da mesma enche e bloqueia a máquina anterior, e assim sucessivamente, até chegar na primeira máquina.

- Em 2, a primeira falha é finalizada e, assim como as máquinas foram parando no início da falha, agora elas vão voltando para o estado *Running*. Quando a última máquina volta a operar, ela começa a esvaziar o *buffer* de entrada e libera a máquina anterior, e assim sucessivamente.
- O ponto 3 marca o momento em que todas as máquinas já estão no estado *Running*.
- No ponto 4 a segunda falha é iniciada na primeira máquina e é possível perceber que as máquinas seguintes vão mudando o estado para *Starved*.
- Porém, no ponto 5, outra falha é iniciada, desta vez na penúltima máquina. Essa terceira falha faz com que as máquinas *pasteurizer* e *first_downstream_machine_1* fiquem bloqueadas (estado *Blocked*) antes de faltar contêineres (estado *Starved*). Como a terceira falha foi curta, assim que a máquina volta a funcionar, ela libera as máquinas bloqueadas e essas voltam para o estado *Running*. Mas, como a segunda falha ainda não terminou, acabam os contêineres e ela vão para o estado *Starved*, que é propagado até a última máquina.
- No ponto 6 a segunda falha é finalizada e, uma por uma, as máquinas vão voltando para o estado *Running*.
- O ponto 7 marca o momento em que todas as máquinas já voltaram para o estado *Running* e estão em operação normal.

A Figura 4.33 apresenta a visão geral dos *buffers* em todo o período do teste. Como as máquinas pararam em diversos momentos, os 10000 ciclos não foram suficientes para processar todas as unidades de entrada, como mostra o segundo gráfico da Figura 4.33.

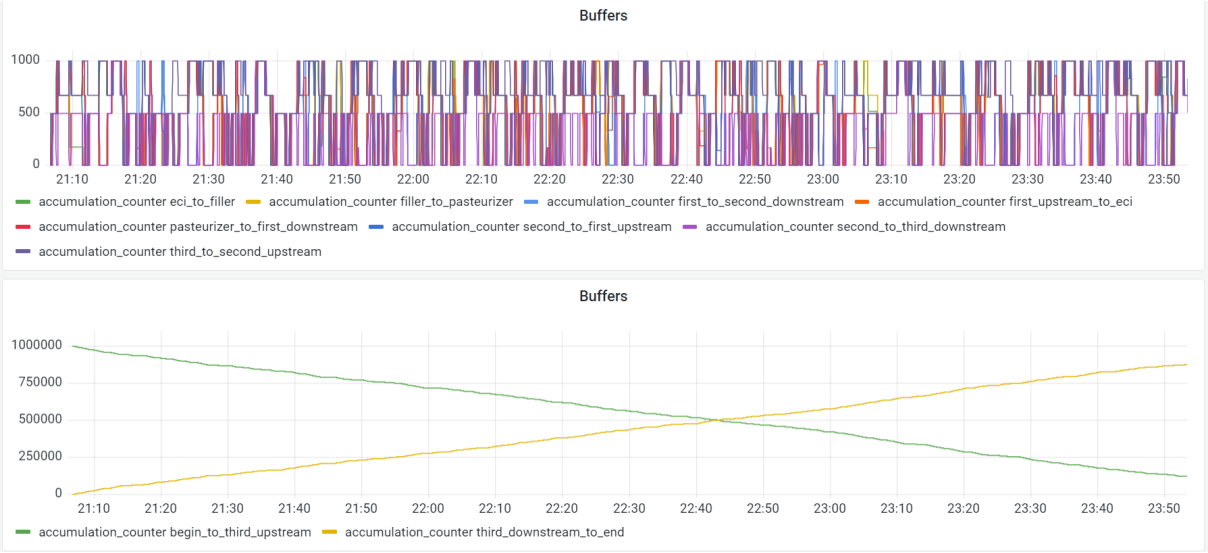

Figura 4.33 – Gráfico com o comportamento dos *buffers* durante todo o Teste 13.

A Tabela 4.21 apresenta o número de vezes que cada máquina esteve em cada um

dos quatro estados possíveis. Como o número de ciclos definido para a simulação não foi suficiente para processar todos os contêineres, a primeira máquina não assumiu o estado *Starved* em nenhum momento. Já a última máquina nunca esteve no estado *Blocked*, isso porque o *buffer* de saída dessa máquina é o último, que é considerado infinito, para poder receber todas as unidades definidas como entrada na simulação.

Tabela 4.21 – Relação da quantidade de vezes que cada máquina esteve em cada estado no Teste 13.

Máquina	**Running**	**Starved**	**Blocked**	**Stopped**
third_upstream_machine_1	140	0	102	38
second_upstream_machine_1	144	37	91	16
first_upstream_machine_1	145	51	71	23
eci_1	146	68	72	6
filler_1	143	69	67	7
pasteurizer	145	71	60	14
first_downstream_machine_1	152	83	62	7
second_downstream_machine_1	150	87	9	55
third_downstream_machine_1	147	138	0	9

4.3.3 Fase 3: Falhas Aleatórias - Teste 14

Como mencionado na Seção 4.3.1, os dados utilizados nos cálculos dos parâmetros MTBF e MTTR são provenientes de uma linha que contém duas enchedoras e duas *first_downstream_machine*. O intuito deste teste é adicionar a segunda máquina nesses dois grupos. A configuração das máquinas está representada na Tabela 4.22, onde é possível perceber que os grupos mencionados contêm duas máquinas com velocidade de 5k CPM. A configuração dos *buffers* e da simulação permanecem iguais as do teste anterior, que são iguais as do Teste 1, apresentadas nas Tabelas 4.2 e 4.3. Os valores dos parâmetros MTBF e MTTR também são os mesmos utilizados no teste anterior, que são os do Cenário 1 apresentados na Tabela 4.20.

Tabela 4.22 – Configuração das máquinas - Teste 14.

Grupo de máquinas	**Quantidade de máquinas habilitadas**	**Tamanho *buffer* interno**	**Velocidade nominal (CPM)**
Third upstream machine	1	1	10000
Second upstream machine	1	1	10000
First upstream machine	1	1	10000
Empty Container Inspector (ECI)	1	1	10000
Filler	2	1	5000
Pasteurizer	1	1	10000
First downstream machine	2	1	5000
Second downstream machine	1	1	10000
Third downstream machine	1	1	10000

A Figura 4.34 apresenta o estado das máquinas e dos *buffers* nos primeiros 30 minutos de simulação. Neste gráfico também é possível perceber que as máquinas assumem

o estado *Stopped*, representado pela cor vermelha, que são as falhas induzidas no sistema.

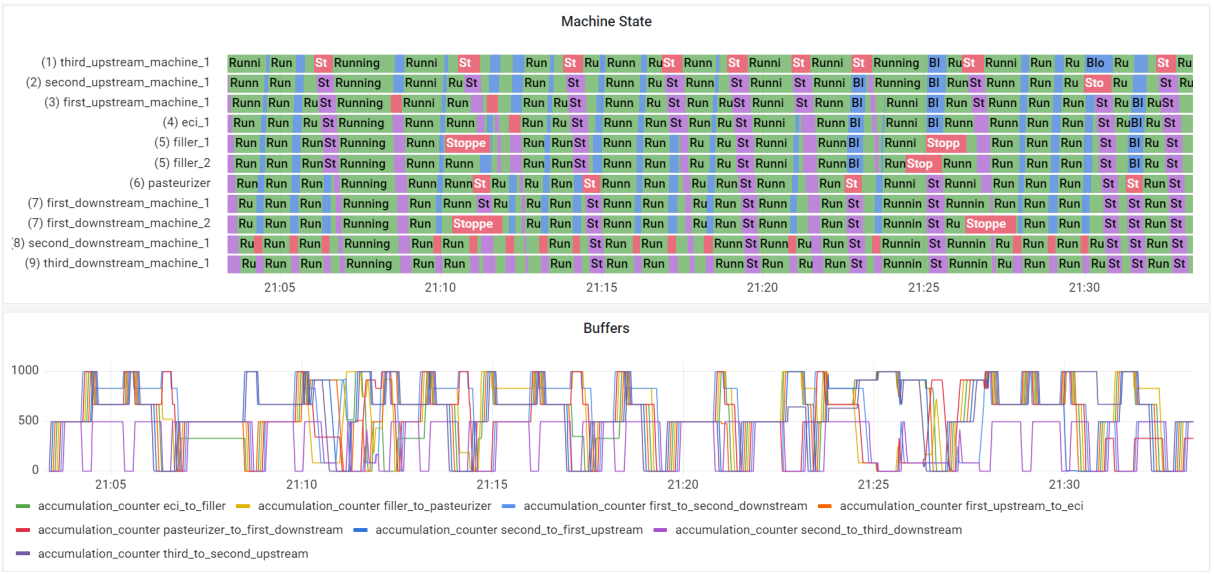

Figura 4.34 – Gráfico do estado das máquinas e do contador dos *buffers* nos primeiros 30 minutos do Teste 14.

A Figura 4.35 apresenta a visão geral dos *buffers* em todo o período do teste. Como as máquinas pararam em diversos momentos, os 10000 ciclos não foram suficientes para processar todas as unidades de entrada, como mostra o segundo gráfico da Figura 4.35.

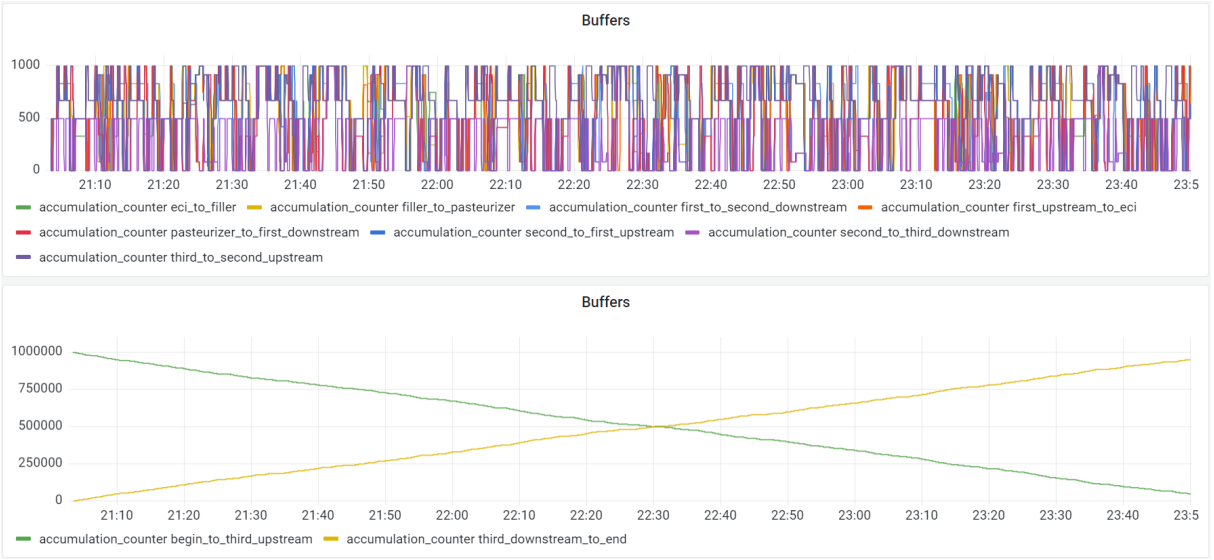

Figura 4.35 – Gráfico com o comportamento dos *buffers* durante todo o Teste 14.

A Tabela 4.23 apresenta o número de vezes que cada máquina esteve em cada um dos quatro estados possíveis. Como as máquinas do mesmo grupo tinham valores diferentes

para o parâmetro MTBF, é normal que a quantidade de vezes em que estiveram no estado *Stopped* seja diferente. Por estarem posicionadas no mesmo lugar na linha, é esperado que máquinas do mesmo grupo apresentem valores semelhantes nos contadores dos demais estados. As variações entre os contadores se dá pelas falhas onde, é possível que, enquanto uma máquina do grupo está no estado *Stopped*, outras máquinas do grupo assumam os outros estados.

Tabela 4.23 – Relação da quantidade de vezes que cada máquina esteve em cada estado no Teste 14.

Máquina	**Running**	**Starved**	**Blocked**	**Stopped**
third_upstream_machine_1	139	0	93	46
second_upstream_machine_1	146	47	86	13
first_upstream_machine_1	151	59	75	17
eci_1	155	72	78	6
filler_1	161	70	85	7
filler_2	160	71	84	6
pasteurizer	158	66	67	26
first_downstream_machine_1	168	95	71	3
first_downstream_machine_2	164	91	66	8
second_downstream_machine_1	170	91	7	73
third_downstream_machine_1	166	160	0	7

4.3.4 Fase 3: Falhas Aleatórias - Teste 15

Este teste é semelhante ao Teste 13, mas, desta vez, os valores do Cenário 2, apresentados na Tabela 4.20, foram utilizados nos parâmetros MTBF e MTTR. As configurações das máquinas, dos *buffers* e da simulação são iguais às do Teste 1, apresentadas nas Tabelas 4.1, 4.2 e 4.3, respectivamente.

A Figura 4.36 apresenta o estado das máquinas e dos *buffers* nos primeiros 30 minutos de simulação. Como os valores do MTBF do Cenário 2 são maiores que os do Cenário 1 para todas as máquinas, é possível notar no gráfico da Figura 4.36 que existem menos falhas na linha quando comparado com os gráficos do Teste 13 e 14. Isso acontece porque o MTBF vai ditar a frequência que as falhas acontecem.

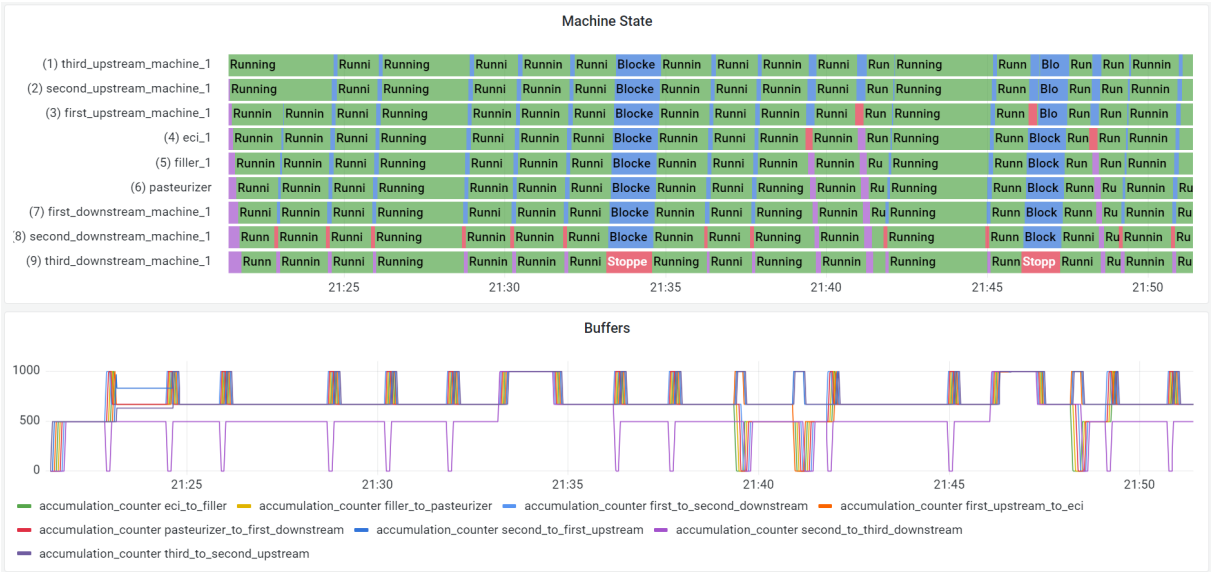

Figura 4.36 – Gráfico do estado das máquinas e do contador dos *buffers* nos primeiros 30 minutos do Teste 15.

A Figura 4.37 apresenta a visão geral dos *buffers* em todo o período do teste. Como as falhas foram menos frequentes, os 10000 ciclos foram suficientes para processar todas as unidades de entrada.

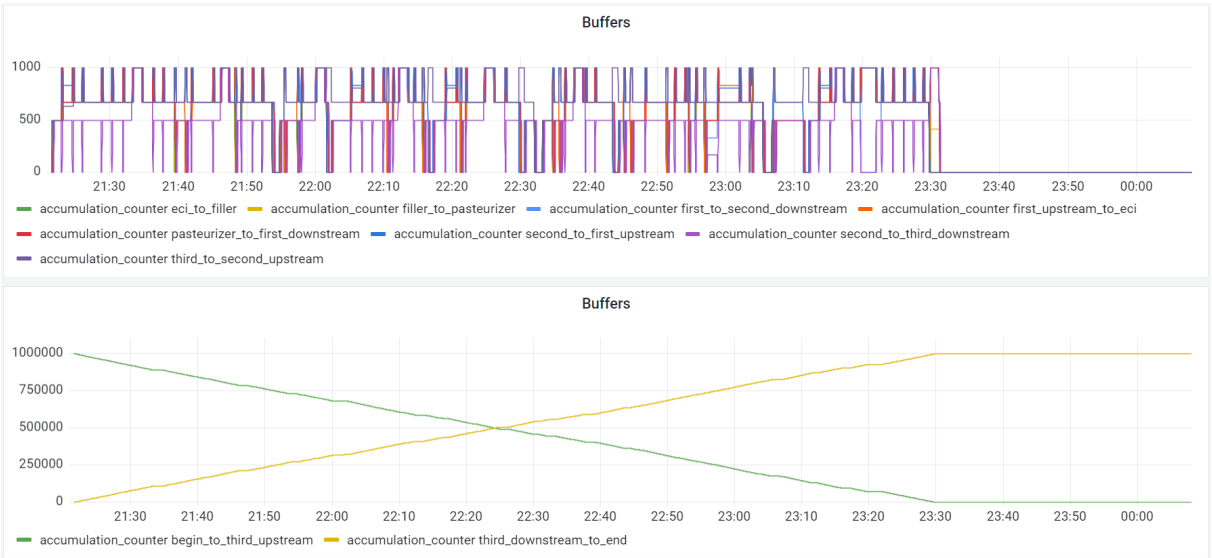

Figura 4.37 – Gráfico com o comportamento dos *buffers* durante todo o Teste 15.

A Tabela 4.24 apresenta o número de vezes que cada máquina esteve em cada um dos quatro estados possíveis. Como o número de ciclos definido para a simulação foi suficiente para processar todos os contêineres, a primeira máquina assumiu o estado *Starved* apenas uma vez, que foi quando o *buffer* de entrada ficou totalmente vazio. Outro

ponto que merece destaque é o contador do *filler_1* em zero para o estado *Stopped*. O valor do MTBF para essa máquina é 14054s, que ultrapassa os 10000 ciclos da simulação, lembrando que cada ciclo é 1s, então a falta não chegou a ser considerada para essa máquina. Situação diferente para o *pasteurizer*, que tem MTBF igual a 2330s, mas que não acionou nenhum falta. Isso é possível porque em todo ciclo de início da falta é sorteado um número inteiro entre 1 e 10, e a falta só acontece quando o número sorteado é maior que 5, conforme apresentado na Seção 3.1.

Tabela 4.24 – Relação da quantidade de vezes que cada máquina esteve em cada estado no Teste 15.

Máquina	**Running**	**Starved**	**Blocked**	**Stopped**
third_upstream_machine_1	71	1	68	3
second_upstream_machine_1	72	5	63	5
first_upstream_machine_1	76	10	56	11
eci_1	76	20	50	7
filler_1	76	27	50	0
pasteurizer	77	27	51	0
first_downstream_machine_1	77	27	50	1
second_downstream_machine_1	78	28	8	43
third_downstream_machine_1	77	70	0	8

4.3.5 Fase 3: Falhas Aleatórias - Teste 16

Finalmente, este teste, assim como o Teste 14, adiciona a segunda máquina no grupo de enchedoras e *first_downstream_machine*. A configuração das máquinas é a mesma do Teste 14, apresentada na Tabela 4.22, onde os grupos mencionados contêm duas máquinas com velocidade de 5k CPM. A configuração dos *buffers* e da simulação permaneceram iguais às do Teste 1, apresentadas nas Tabelas 4.2 e 4.3. Os valores dos parâmetros MTBF e MTTR são os mesmos utilizados no teste anterior, que são os do Cenário 2 apresentados na Tabela 4.20.

A Figura 4.38 apresenta o estado das máquinas e dos *buffers* nos primeiros 30 minutos de simulação. Assim como no teste anterior, é possível notar no gráfico que existem menos falhas na linha quando comparado com os gráficos do Teste 13 e 14. Isso acontece pois os valores do MTBF do Cenário 2 são maiores que os do Cenário 1 para todas as máquinas e é esse parâmetro que define a frequência que as falhas acontecem.

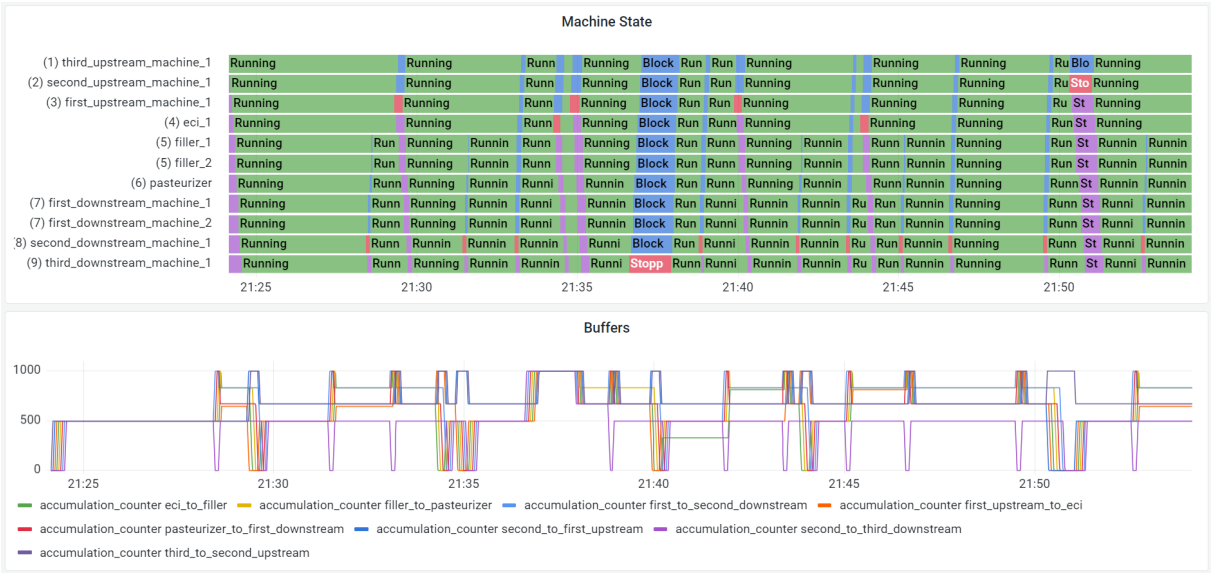

Figura 4.38 – Gráfico do estado das máquinas e do contador dos *buffers* nos primeiros 30 minutos do Teste 16.

A Figura 4.39 apresenta a visão geral dos *buffers* em todo o período do teste. Como as falhas foram menos frequentes, os 10000 ciclos foram suficientes para processar todas as unidades de entrada.

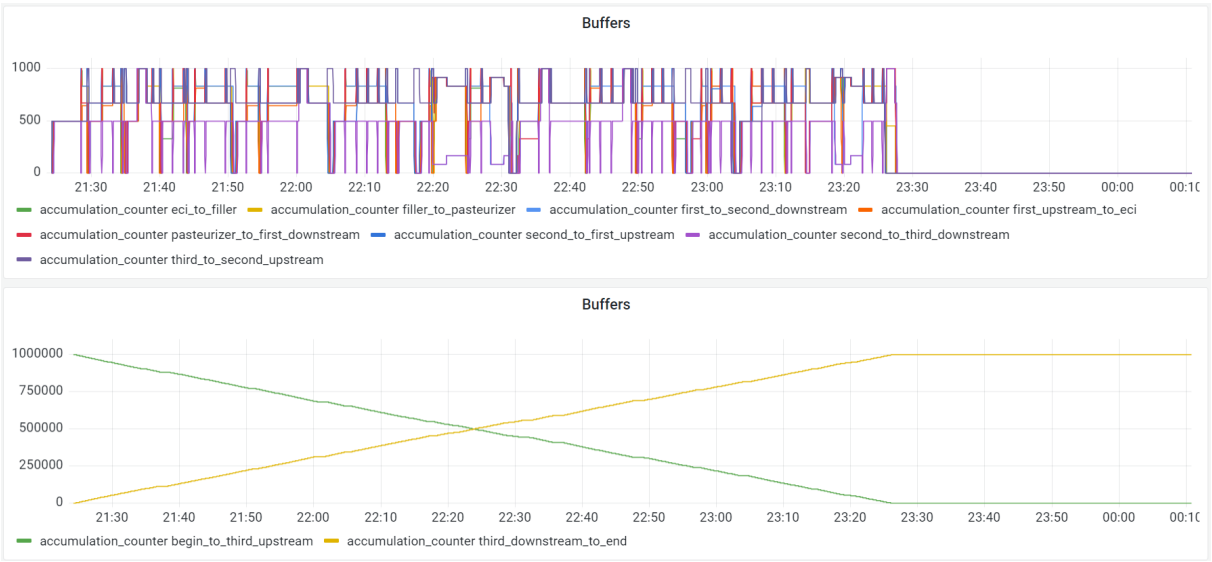

Figura 4.39 – Gráfico com o comportamento dos *buffers* durante todo o Teste 16.

A Tabela 4.25 apresenta o número de vezes que cada máquina esteve em cada um dos quatro estados possíveis. Como o número de ciclos definido para a simulação foi suficiente para processar todos os contêineres, a primeira máquina assumiu o estado *Starved* apenas uma vez, que foi quando o *buffer* de entrada ficou totalmente vazio.

O contador do *filler_1* em zero para o estado *Stopped* se justifica pelo mesmo motivo mecionado no teste anterior, como o valor do MTBF para essa máquina é 14054s, ele ultrapassa os 10000 ciclos da simulação, então a falta não chegou a ser considerada para essa máquina. Já o *filler_2*, que tem MTBF igual a 1636s, a situação é a mesma mencionada para o *pasteurizer* no teste anterior. Apesar do MTBF não ultrapassar o número de ciclos, a falta não foi acionada em nenhum momento.

Tabela 4.25 – Relação da quantidade de vezes que cada máquina esteve em cada estado no Teste 16.

Máquina	**Running**	**Starved**	**Blocked**	**Stopped**
third_upstream_machine_1	52	1	50	2
second_upstream_machine_1	53	4	46	4
first_upstream_machine_1	54	8	36	11
eci_1	56	19	32	6
filler_1	68	24	45	0
filler_2	68	24	45	0
pasteurizer	70	24	46	1
first_downstream_machine_1	71	24	46	2
first_downstream_machine_2	70	24	46	1
second_downstream_machine_1	70	24	5	42
third_downstream_machine_1	70	66	0	5

4.4 Considerações

Neste capítulo foram apresentados os testes de validação do PLSIM para linhas simples e com ramificação. Essa etapa foi importante para avaliar melhor o funcionamento da lógica e validar o funcionamento. Os cenários de teste foram escolhidos com o intuito de abranger diferentes comportamentos, mudando a velocidade das máquinas em diferentes pontos da linha. Conforme apresentado, em alguns testes, os 10000 ciclos não foram suficientes para processar totalmente 1000000 de contêineres de entrada. Contudo, esse fato não comprometeu a análise dos resultados, e os testes não foram refeitos por dois motivos: (1) aumentar o número de ciclos resultaria apenas em um prolongamento do tempo que o sistema fica no mesmo estado até que todos os contêineres fossem processados, sem oferecer benefícios significativos para validação do cenário de teste; e (2) optou-se por manter número de ciclos constante em todos os testes, visando facilitar a comparação dos resultados.

Foram identificados alguns erros que foram completamente corrigidos. Os erros identificados na primeira fase de testes foram herdados do simulador base, já os erros apresentados na segunda seção estão relacionados com a forma como o PLSim foi implementado. Foram apresentados também os resultados dos testes com falhas, que foi

importante para adicionar uma característica aleatória na linha, deixando assim a simulação mais parecida com os sistemas reais.

5 Conclusão e Perspectivas

Conforme apresentado no Capítulo 1, impulsionados pela Indústria 4.0 e a IIoT, as indústrias têm investido em novas tecnologias para digitalizar e conectar de ponta a ponta suas plantas. Dessa forma, é possível ter maior visibilidade do funcionamento das linhas de produção para propor lógicas de controle mais adequadas e obter melhores resultados. Em uma dessas inciativas, surgiu a necessidade da criação de uma ferramenta capaz de simular uma linha de envase para viabilizar testes mais assertivos de novas tecnologias ainda na etapa de desenvolvimento, com o intuito de identificar possíveis falhas antes de serem aplicadas nas linhas reais. Essa demanda surgiu no contexto de um projeto que está em contínua expansão de funcionalidades e instalação em novas linhas, aumentando ainda mais a necessidade de se ter uma ferramenta como essa. Este trabalho apresentou as etapas de validação de um simulador de linhas de envase de cerveja para suprir essa demanda.

O simulador proposto nesta dissertação, o PLSim, apresentou os resultados esperados. O objetivo principal foi cumprido, que consistia em desenvolver uma ferramenta de simulação que representasse o comportamento de uma linha de envase real. Os testes realizados comprovam o funcionamento da lógica que rege a simulação tanto para linhas simples quanto para linhas com ramificações. Além disso, também foi validada a funcionalidade de paradas aleatórias, que simulam as falhas das máquinas. A linha de envase a ser utilizada na simulação é configurável, o que faz com que ele seja flexível e possa ser utilizado em diferentes configurações de linha. O PLSim foi desenvolvido utilizando apenas recursos *open-source*, o que facilita a instalação e compartilhamento da ferramenta.

Há espaço para implementação de melhorias, a fim de que o simulador fique cada vez mais próximo do comportamento de uma linha de envase real. Um exemplo é a adição de porcentagens de rejeição em alguns pontos da linha. Nos sistemas reais existem máquinas inspetoras de contêineres, que retiram da linha aqueles que não estão dentro dos padrões esperados, como exemplo, uma garrafa quebrada. Além de melhorias na lógica, é possível desenvolver recursos para facilitar a utilização do simulador, com o desenvolvimento de uma interface para configuração e monitoramento da linha.

5.1 Trabalhos Futuros

Esta seção apresenta os próximos passos para continuidade do desenvolvimento do PLSim, a fim de que deixe-o mais robusto e completo. O primeiro ponto é a implementação da integração com um servidor OPC UA. O simulador, assim como as linhas

de envase real, deve disponibilizar suas informações em um servidor OPC UA, pois é de lá que as outras aplicações irão lê-las. Essa integração deve permitir que o PLSim atualize continuamente o valor de suas variáveis no servidor, como estado e velocidade das máquinas, acumulação dos *buffers*, entre outros. O PLSim também deve ser capaz de ler dados do servidor que foram escritas por outros sistemas e, mediante alteração no código, utilizá-la na simulação. Essa funcionalidade é importante para utilização do simulador com o PaCo, que irá escrever o *setpoint* de velocidade para as enchedoras. O PLSim deve ser capaz de ler o *setpoint* e aplicá-lo na velocidade da máquina.

Atualmente, conforme apresentado no Capítulo 3, a configuração da linha de envase a ser utilizada na simulação é feita em um dos arquivos do código, o que não é intuitivo para muitas pessoas. Com o objetivo de tornar a ferramenta mais amigável e de fácil utilização, é necessário desenvolver uma interface visual que permita a configuração da linha e da simulação. Pode-se pensar em um banco de dados para salvar várias configurações de linhas, de modo que a linha desejada seja selecionada antes do início da simulação. Além de telas de configuração, pretende-se desenvolver uma tela para monitoramento da linha durante a simulação.

Referências

ACHKAR, V. G. et al. A simulation-based tool to support decision-making in logistics design of a can packaging line. *International Journal of Food Engineering*, v. 16, n. 5-6, p. 20170089, 2020.

AnyLogic Company. *AnyLogic simulation software*. 2024. Acesso em: 13/08/2024. Disponível em: <https://www.anylogic.com/>.

Autodesk Company. *FlexSim*. 2024. Acesso em: 13/08/2024. Disponível em: <https://www.flexsim.com/>.

BABAYIGIT, B.; ABUBAKER, M. Industrial internet of things: A review of improvements over traditional scada systems for industrial automation. *IEEE Systems Journal*, v. 18, n. 1, p. 120–133, 2024.

Borsato Pinhão, J. da M. O.; IGNACIO, A. A. V.; COELHO, O. An integer programming mathematical model with line balancing and scheduling for standard work optimization: A realistic application to aircraft engines assembly lines. *Computers Industrial Engineering*, v. 173, p. 108652, 2022. ISSN 0360-8352.

BOYSEN, N.; SCHULZE, P.; SCHOLL, A. Assembly line balancing: What happened in the last fifteen years? *European Journal of Operational Research*, v. 301, n. 3, p. 797–814, 2022. ISSN 0377-2217.

Calış Duman, M.; AKDEMIR, B. A study to determine the effects of industry 4.0 technology components on organizational performance. *Technological Forecasting and Social Change*, v. 167, p. 120615, 2021. ISSN 0040-1625.

C.E. O. O. OBIUKWU, O. G. O. Reduction of machine breakdown in the brewery manufacturing company, a case study of brewery “a” in the eastern nigeria. *Miscellaneous*, v. 07, n. 03, p. 65–74, 2023.

DEREJE, G.; ABEYA, G.; GOPAL, M. Manufacturing system modeling and performance analysis of mineral water production line using arena simulation. *International Journal of Engineering and Advanced Technology*, v. 9, p. 2249–8958, 06 2020.

DHIMAN, P.; KUMAR, A. A situational based reliability indices estimation of ult freezer using preventive maintenance under fuzzy environment. *Miscellaneous*, v. 8, n. 3, p. 477–503, 2023.

Docker. *Docker*. 2024. Acesso em: 29/08/2024. Disponível em: <https://www.docker.com/>.

FORBUS, J. J.; BERLEANT, D. Discrete-event simulation in healthcare settings: A review. *Modelling*, v. 3, n. 4, p. 417–433, 2022. ISSN 2673-3951. Disponível em: <https://www.mdpi.com/2673-3951/3/4/27>.

Inductive Automation. *Ignition*. 2024. Acesso em: 18/08/2024. Disponível em: <https://inductiveautomation.com/ignition/>.

InfluxData. *InfluxDB*. 2024. Acesso em: 29/08/2024. Disponível em: <https://www.influxdata.com/>.

JaamSim. *JaamSim*. 2024. Acesso em: 13/08/2024. Disponível em: <https://jaamsim.com/>.

KLIMENT, M. et al. Production efficiency evaluation and products' quality improvement using simulation. *International Journal of Simulation Modelling*, v. 19, p. 470–481, 09 2020.

LANG, S. et al. Open-source discrete-event simulation software for applications in production and logistics: An alternative to commercial tools? *Procedia Computer Science*, v. 180, p. 978–987, 2021. ISSN 1877-0509. Proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020).

LAURINDO, Q. M. G. et al. Comissionamento de uma linha de produção no software de simulação a eventos discretos ururau. *Revista Gestão da Produção Operações e Sistemas*, v. 12, n. 1, p. 167, mar. 2017.

LIN, F. et al. State estimation of multichannel networked discrete event systems. *IEEE Transactions on Control of Network Systems*, v. 7, n. 1, p. 53–63, 2020.

MICHLOWICZ, E. Assessment of the modernized production system through selected tpm method indicators. *Eksploatacja i Niezawodność – Maintenance and Reliability*, v. 24, n. 4, p. 677–686, 2022. ISSN 1507-2711. Disponível em: <https://doi.org/10.17531/ein.2022.4.8>.

Microsoft. *SQL Server 2022*. 2024. Acesso em: 29/08/2024. Disponível em: <https://www.microsoft.com/pt-br/sql-server/sql-server-2022>.

National Institute of Standards and Technology - NIST. *Why is UTC used as the acronym for Coordinator Universal Time instead of CUT?* 2024. Acesso em: 12/11/2024. Disponível em: <https://www.nist.gov/pml/time-and-frequency-division/how-utcnist-related-coordinated-universal-time-utc-international>.

NICOLETTI, A. et al. Simulação de eventos discretos para análise da disponibilidade fabril em uma linha de envase de bebidas. *Revista de Ciência Tecnologia*, v. 19, p. 19–29, 06 2016.

Observatório da Embalagem. *Observatório: Sutilezas a observar na mudança de embalagens clássicas*. 2018. Acesso em: 18/08/2024. Disponível em: <https://embalagemmarca.com.br/2018/05/sutilezas-a-observar-na-mudanca-de-embalagens-classicas/>.

OPC Foundation. *Unified Architecture*. 2024. Acesso em: 29/08/2024. Disponível em: <https://opcfoundation.org/about/opc-technologies/opc-ua/>.

PostgreSQL. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. 2024. Acesso em: 29/08/2024. Disponível em: <https://www.postgresql.org/>.

PRADO, A. V.; MENDONçA, A. R.; DUTRA, M. D. d. S. SimulaÇÃo de uma linha de produÇÃo de bebidas em uma indÚstria no estado de goiÁs. *Anais do Congresso Internacional de Engenharia Mecânica e Industrial - 23º CONEMI (Vitória-ES)*, 2023.

ProdSim. *ProdSim*. 2024. Acesso em: 13/08/2024. Disponível em: <https://prodsim.readthedocs.io/en/latest/>.

Python. *Python*. 2024. Acesso em: 29/08/2024. Disponível em: <https://www.python.org/>.

Rockwell Automation. *Arena Simulation Software*. 2024. Acesso em: 13/08/2024. Disponível em: <https://www.rockwellautomation.com/en-us/products/software/arena-simulation.html>.

Salabim. *Salabim*. 2024. Acesso em: 13/08/2024. Disponível em: <https://www.salabim.org/>.

SANTOS, R. S. d.; ALENCAR, M. A. F. d. Cadeia de markov aplicada a itub4. *Peer Review*, v. 6, n. 8, p. 295–307, abr. 2024. Disponível em: <https://peerw.org/index.php/journals/article/view/2097>.

SIEMENS. *Tecnomatix Plant Simulation*. 2024. Acesso em: 13/08/2024. Disponível em: <https://plm.sw.siemens.com/en-US/tecnomatix/products/plant-simulation-software/?pk_vid=81d9918dbc451fd1c51c8ef045c01a1e1723595134c9d484>.

Simio. *The Simio Discrete Event Simulation Platform*. 2024. Acesso em: 18/11/2024. Disponível em: <https://www.simio.com/>.

UJAM, C. J.; GODWIN, H. C. Optimization of packaging line performance: A case study of ab breweries in nigeria. *Journal of Engineering Research and Reports*, v. 1, n. 1, p. 1–22, Apr. 2018.

Zegla. *Linha Compacta de Latas*. 2024. Acesso em: 22/11/2024. Disponível em: <https://www.zegla.com.br/equipamento/linha-compacta-de-latas,79>.

ZENG, P.; SHAO, W.; HAO, Y. Study on preventive maintenance strategies of filling equipment based on reliability-cantered maintenance. *Miscellaneous*, v. 28, n. 2, p. 689–697, 2021.