UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Instituto de Ciência e Tecnologia de Sorocaba
Campus de São João da boa vista

LUCAS BORGES RODRIGUES DE SÁ

PID 4.0: CONTROLADOR REDUNDANTE DE PROCESSOS PARA A INDÚSTRIA 4.0 BASEADO EM MICROSSERVIÇOS

Sorocaba
2023

LUCAS BORGES RODRIGUES DE SÁ

PID 4.0: CONTROLADOR REDUNDANTE DE PROCESSOS PARA A INDÚSTRIA 4.0 BASEADO EM MICROSSERVIÇOS

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica, junto ao Programa de Pós-Graduação em Engenharia Elétrica, Interunidades, entre o Instituto de Ciência e Tecnologia de Sorocaba e o Campus de São João da Boa Vista da Universidade Estadual Paulista "Júlio de Mesquita Filho".

Área de concentração: Automação
Orientador: Profº Dr. Eduardo Paciência Godoy

Sorocaba
2023

S111p

Sá, Lucas Borges Rodrigues de
PID 4.0: Controlador Redundante de Processos para a Indústria 4.0 Baseado em Microsserviços / Lucas Borges Rodrigues de Sá. -- Sorocaba, 2023
67 p. : il., tabs., fotos

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Ciência e Tecnologia, Sorocaba
Orientador: Eduardo Paciência Godoy

1. Arquitetura Orientada a Serviços. 2. Redundância de Controladores. 3. Sistemas de Controle via rede. 4. Internet das Coisas Industrial. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Ciência e Tecnologia, Sorocaba. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Sorocaba

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: PID 4.0: Controlador Redundante de Processos para a Indústria 4.0 Baseado em Microsserviços

AUTOR: LUCAS BORGES RODRIGUES DE SÁ
ORIENTADOR: EDUARDO PACIÊNCIA GODOY

Aprovado como parte das exigências para obtenção do Título de Mestre em Engenharia Elétrica, área: Automação pela Comissão Examinadora:

Prof. Dr. EDUARDO PACIÊNCIA GODOY (Participaçao Presencial)
Departamento de Engenharia de Controle e Automacao / Instituto de Ciência e Tecnologia - UNESP - Câmpus de Sorocaba

unesp
UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
Prof. Eduardo Paciencia Godoy
eduardo.godoy@unesp.br
ICPEdu - Certificado Pessoal
Assinado de forma digital por Eduardo Paciencia Godoy:29137583808
Dados: 2023.02.10 16:08:52 -03'00'

Prof. Dr. EDUARDO VERRI LIBERADO (Participaçao Presencial)
Departamento de Engenharia de Controle e Automacao / Instituto de Ciência e Tecnologia - UNESP - Câmpus de Sorocaba

Eduardo Verri Liberado:33018491807
Assinado de forma digital por Eduardo Verri Liberado:33018491807
Dados: 2023.02.10 16:03:43 -03'00'

Prof. Dr. GUILHERME SERPA SESTITO (Participaçao Virtual)
Departamento Acadêmico de Elétrica - CP - DAELE-CP / Universidade Tecnológica Federal do Paraná – Campus Cornélio Procópio

GUILHERME SERPA SESTITO:34604351864
Assinado de forma digital por GUILHERME SERPA SESTITO:34604351864
Dados: 2023.02.10 15:49:46 -03'00'

Sorocaba, 10 de fevereiro de 2023

LUCAS BORGES RODRIGUES DE SÁ

PID 4.0: CONTROLADOR REDUNDANTE DE PROCESSOS PARA A INDÚSTRIA 4.0 BASEADO EM MICROSSERVIÇOS

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica, junto ao Programa de Pós-Graduação em Engenharia Elétrica, Interunidades, entre o Instituto de Ciência e Tecnologia de Sorocaba e o Campus de São João da Boa Vista da Universidade Estadual Paulista "Júlio de Mesquita Filho".

Comissão Examinadora

Profº Dr. Eduardo Paciência Godoy
UNESP – Instituto de Ciência e Tecnologia de Sorocaba
Orientador

Profº Dr. Guilherme Serpa Sestito
UTFPR - Universidade Tecnológica Federal do Paraná

Profº Dr. Eduardo Verri Liberado
UNESP – Instituto de Ciência e Tecnologia de Sorocaba

Sorocaba
2023

"O preço da liberdade é a eterna vigilância"

Thomas Jefferson

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus por me permitir mais essa conquista em minha vida.

Aos meus pais pela educação, carinho e valores que fizeram a diferença em minha vida me deixando preparado para assumir novos desafios e conquista-los.

A minha esposa e filhas pelo amor incondicional e compreensão nos momentos difíceis que a vida nós apresenta, sem isso não seria possível se levantar quando caímos.

A Marinha do Brasil por ser uma instituição que acredita e valoriza sua força de trabalho investindo em seu desenvolvimento pessoal e técnico.

Ao meu orientador Prof. Dr. Eduardo Paciência Godoy por ter sabedoria em compartilhar seu vasto conhecimento, por sua paciência e dedicação.

Ao apoio na realização dessa pesquisa pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processo nº 2018/19984-4.

RESUMO

A Indústria 4.0 iniciou um processo de digitalização e para isso vêm utilizando novas tecnologias como a Internet das Coisas, Computação em Nuvem, Arquitetura Orientada a Serviços e Sistemas de Controle em Rede. O uso integrado dessas tecnologias em aplicações recentes de automação e controle de processos proporcionam vantagens, como a interoperabilidade vertical e a padronização da comunicação, através do compartilhamento de serviços em rede. Redundância, na automação industrial, significa manter sistemas duplicados ou triplicados para garantir a disponibilidade de processos e dispositivos críticos. Tendo em vista o contexto da Indústria 4.0 este trabalho desenvolveu uma nova abordagem para se fazer redundância de controladores de processo na indústria. O objetivo foi desenvolver um controlador redundante usando uma arquitetura de automação orientada a microsserviços. O controlador redundante PID4.0 é baseado em uma modificação do algoritmo PIDPlus para controle de processos em rede e operação como um microsserviço. Adicionalmente, uma aplicação de redundância (orquestrador) é responsável pela orquestração dos microsserviços e o registro das variáveis de controle, permitindo o compartilhamento de dados das variáveis de processo entre os controladores redundantes (em réplica). O PID 4.0 pode ser alocado em diferentes *hardwares*, no computador ou até mesmo em nuvem industrial, fornecendo versatilidade a este tipo de solução. O projeto foi desenvolvido em uma planta piloto de controle de processos industriais orientada a microsserviços. Resultados experimentais relacionados ao desempenho de controle e ao tempo de resposta da malha de controle em diferentes cenários demonstram a viabilidade e robustez da implementação do controlador PID4.0 redundante como um serviço. Adicionalmente, um estudo do impacto da forma de comunicação entre os serviços e controladores redundantes no desempenho de controle foi realizado. Essa abordagem tem uma promissora redução de custos e tempo de desenvolvimento das aplicações utilizando a virtualização de controladores, além de redução de *hardware* utilizado.

PALAVRAS-CHAVE: Indústria 4.0. Sistemas de Controle. Arquitetura Orientada a Serviços. Redundância de controlador. Hardware.

ABSTRACT

Industry 4.0 starts the digitalization process and for this they have been using new technologies as Industrial Internet of Things (IIoT), Cloud Computing, Service Oriented Architectures and Networked Control Systems. The integrated use of these technologies in recent applications of automation and process control provide advantages, such as vertical interoperability and standardization of communication, through the sharing of networked services. Redundancy, in the automation industry, means keeping systems duplicate or triplicate in order to ensure availability for processes and critical devices. In view of the Industry 4.0 context, this work has developed a new approach to making redundancy of process controllers in the industry. The goal was to develop a redundant controller using a microservice-oriented automation architecture. The redundant controller (PID4.0) is based on modification of the PIDPlus algorithm for network process control and operation such as a microservice. Additionally, a redundancy application (orchestrator) is responsible for orchestrating microservices and recording control variables, allowing the sharing of data from process variables among redundant controllers (in replica). PID 4.0 can be allocated on different hardware, on the computer, or even on industrial cloud, providing versatility to this type of system. The project was developed in a microservice-oriented industrial process control pilot plant. Experimental results related to control performance and control mesh response time in different scenarios demonstrate the feasibility and robustness of implementing the redundant PID4.0 controller as a service. Additionally, a study of the impact of the way of communication between the services and the redundant controllers on control performance was carried out. This approach has a promising reduction in costs and development time of applications using controller virtualization, in addition to hardware reduction utilized.

KEYWORDS: Industry 4.0. Control Systems. Service Oriented Architectur. Controller Redundancy. Hardware.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface*
CLP	*Controlador Lógico Programável*
CPS	*Cyber-physical system*
DAQ	*Data Acquisition*
DPWS	*Device Profile for Web Services*
HTTP	*Hyper Text Transfer Protocol*
HTTPS	*Hyper Text Transfer Protocol Secure*
I2C	*Inter Integrated Circuit*
I4.0	*Industry 4.0*
IIoT	*Industrial Internet of Things*
IoT	*Internet of Things*
IP	*Internet Protocol*
ISA	*International Society of Automation*
JSON	*JavaScript Object Notation*
MOA	*Microservice Oriented Architecture*
MQTT	*Message Queuing Telemetry Transport*
MV	*Manipulated Variable*
NATS	*Messaging System*
NPM	*Node Package Manager*
PID	*Proportional Integral Derivative*
PLC	*Programmable Logic Controller*
PV	*Process Variable*

REST	*Representational State Transfer*
RPC	*Remote Procedure Call*
SCADA	*Supervisory Control and Data Acquisition*
SOA	*Service Oriented Architecture*
TCP	*Transmission Control Protocol*
TA	*Tecnologia da Automação*
TI	*Tecnologia da Informação*
URL	*Uniform Resource Locator*

SUMÁRIO

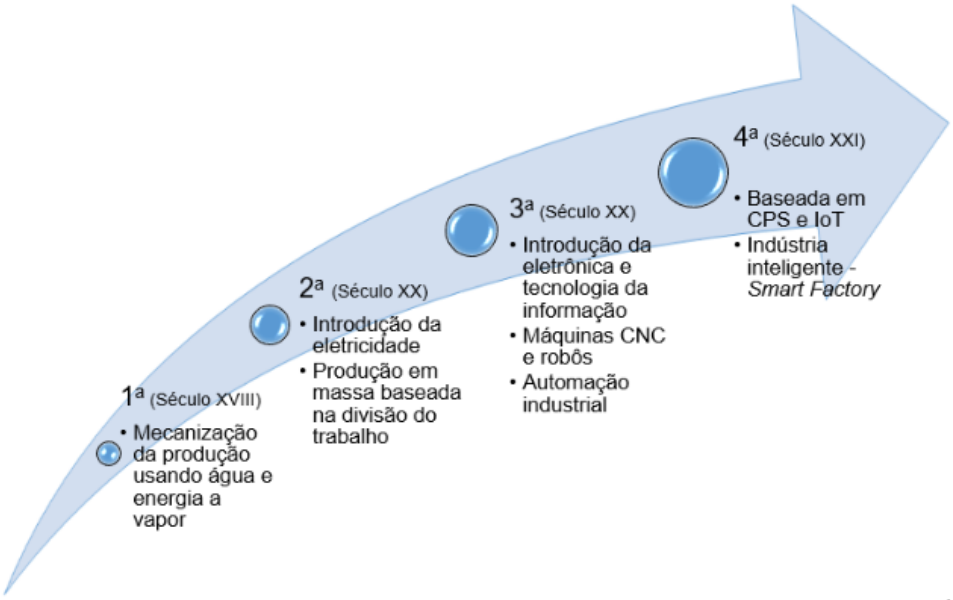
1 INTRODUÇÃO E JUSTIFICATIVA

A automação industrial está evoluindo rapidamente chegando em uma nova revolução chamada Indústria 4.0, e para isso vem integrando novas tecnologias com o intuito de aumentar a eficiência e produtividade, com a convergência entre tecnologias da automação (TA) e tecnologia da informação (TI). Para melhorar os processos industriais é fundamental utilizar tecnologias de automação, que com sua alta capacidade em acelerar processos de manufatura e produção, obtém maior eficiência e qualidade com menores custos e tempos. Dessa forma, a indústria tem evoluído de forma a incorporar novas tecnologias e consequentemente obter maior produtividade (Sauter et al., 2011; Bangemann et al., 2014).

O desenvolvimento e evolução das tecnologias digitais contribuíram para a criação de novos métodos de produção industrial baseados na automação, robótica, inteligência artificial,Internet das Coisas e inteligência de dados, dentre outras inovações. Dentro da indústria a utilização dessas tecnologias de forma coordenada a fim de aumentar a competitividade dos negócios, otimizar a eficiência da cadeia produtiva, agregar valor ao produto, utilizar os recursos de forma mais sustentável e customizar as soluções tecnológicas é chamada de Indústria 4.0.

Chamada de quarta revolução industrial a I4.0, é um novo conceito que representa uma evolução dos sistemas produtivos atuais a partir da convergência entre TA e TI. (BLANCHET et al., 2014; Lu, 2017) como pode ser visto na Figura 1.

Figura 1 – As Revoluções Industriais

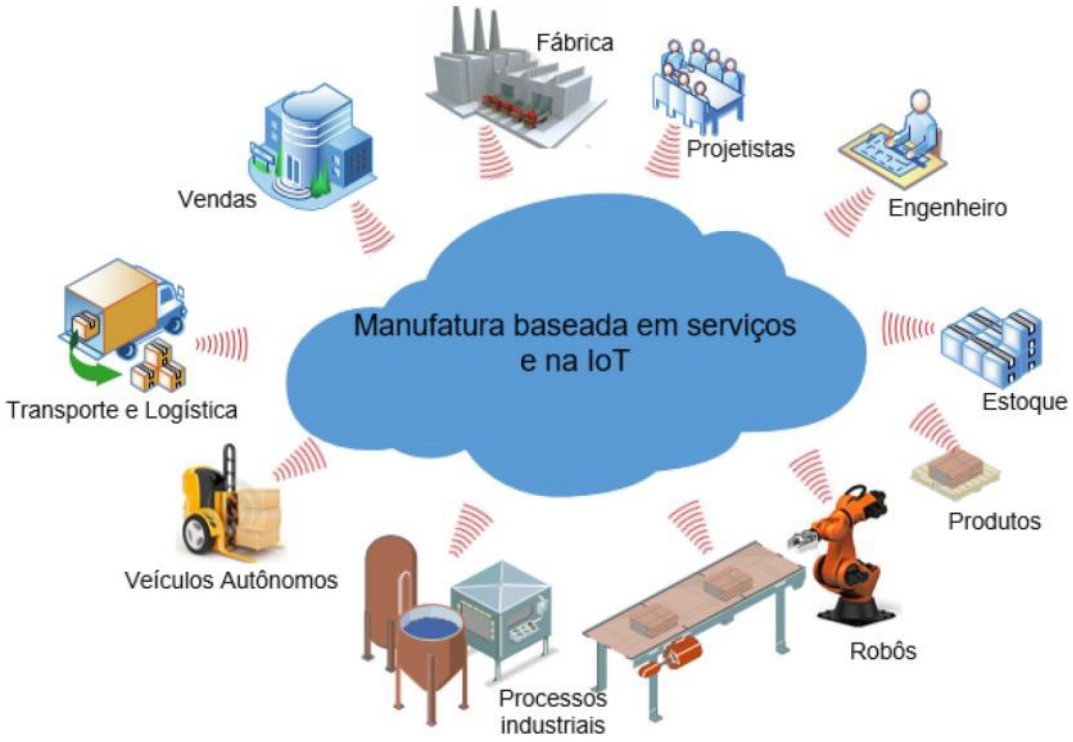


Fonte: Pisching (2017).

A integração entre todas essas tecnologias é o grande desafio da I4.0, tendo como meta uma

nova realidade produtiva, onde tudo estará conectado permitindo melhores decisões de produção, custo e segurança, tudo sob demanda em tempo real. Neste sentido, diversos trabalhos são desenvolvidos buscando soluções que agreguem essas tecnologias e permitam a integração dos processos produtivos e equipamentos de automação a serviços de TI que estão armazenados na nuvem, conforme Figura 2.

Figura 2 – Integração de Processos na Industria 4.0 através de Tecnologias de IoT e Serviços.

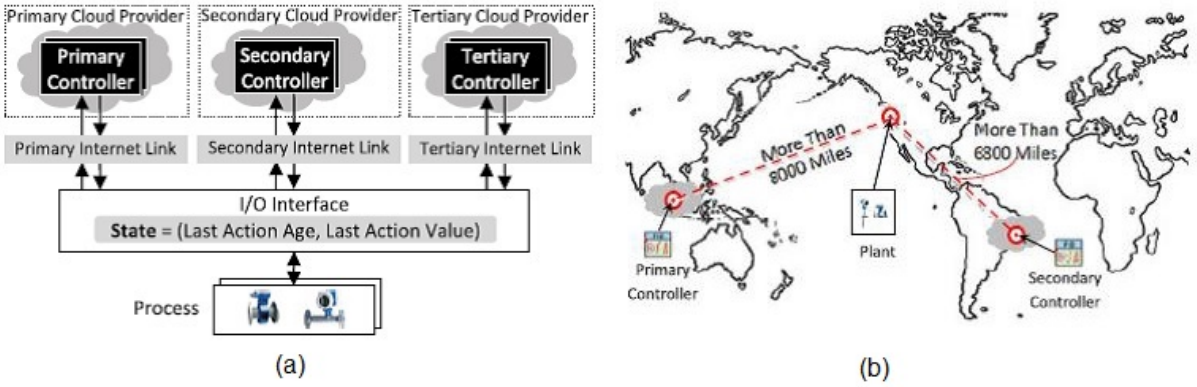


Fonte: Pisching (2018).

Um novo serviço em nuvem é defendido por Hegazy e Heffeda (2015), "automação industrial como serviço". Conforme apresentado na Figura 3 é proposto pelos autores colocar dois controladores hospedados em nuvem e fisicamente separados. Com essa proposta traria inúmeros benefícios, como redução de custos e tempo de desenvolvimento das aplicações utilizando a virtualização de controladores, além de redução de *hardware* utilizado. É apresentado uma implementação de redundância em nuvem de controladores industriais atuando no controle de uma planta utilizando a nuvem comercial *Amazon Web Services*.

Delsing (2017) apresenta uma Arquitetura Orientada a Serviços (SOA) como infraestrutura de integração e comunicação, utilizando o conceito de nuvem em uma rede local, desta forma os componentes e dispositivos de automação são disponibilizados como serviços, provendo interoperabilidade total entre as aplicações industriais e desta forma propõe um modelo de arquitetura de automação em nuvem local para substituir o antigo ISA-95, trazendo maior flexibilidade entre os dispositivos IoT (Internet of Things).

Figura 3 – (a) Redundância de Controladores, (b) Localização de Controladores em Nuvem.



Fonte: Hegazy e Hefeeda (2015).

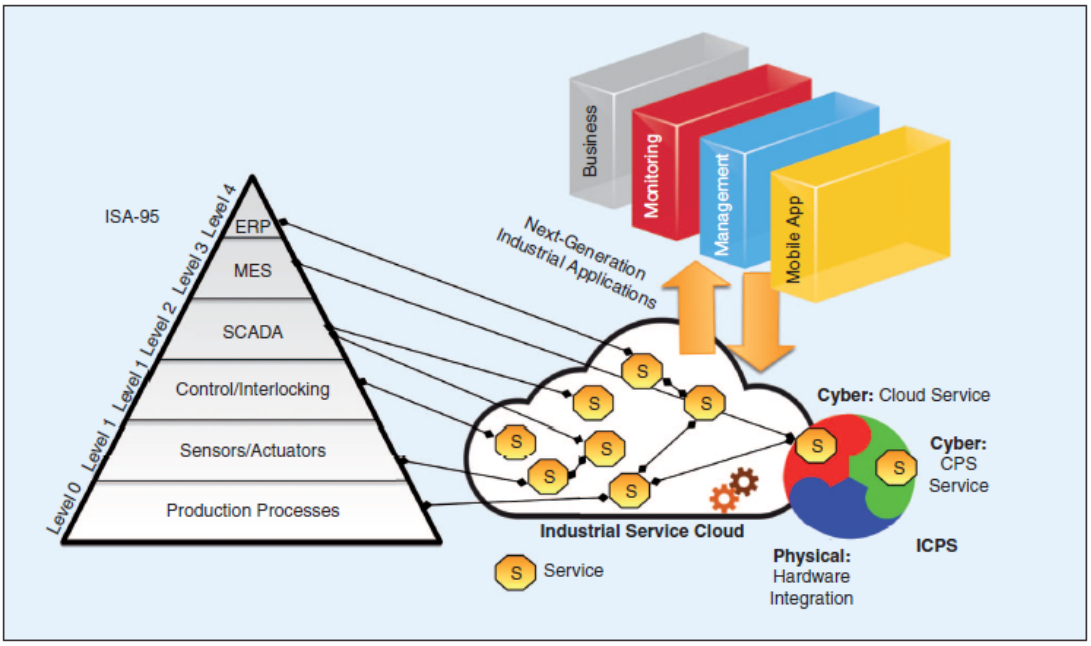
Por fim Colombo et al. (2017) afirmam que através do SOA um *hardware* programável tem grandes chances de se tornar um serviço virtualizado, o que permitiria sua configuração ou programação por aplicativos. Sendo assim, as funcionalidades dos dispositivos podem ser operadas por diferentes módulos, com sistemas independentes da heterogeneidade do *hardware/software*, permitindo a interoperabilidade dos sistemas como mostrado na Figura 4. É possível observar nessa figura que novos serviços poderão ser criados a partir de serviços que já existem utilizando o método de orquestração ou coreografia, também é possível observar que com isso teremos uma migração da arquitetura ISA-95 tradicional para uma arquitetura SOA em nuvem mais flexível onde todos os recursos, sistemas e equipamentos são disponibilizados em uma mesma infraestrutura em nuvem de forma padronizada e interoperável.

Diante deste contexto da I4.0, buscou-se uma forma de utilizar essas tecnologias em sistemas de redundância em aplicações de automação e controle de processos que tem fundamental importância na indústria. Para aplicações críticas, uma falha que implique na parada da planta pode resultar em um considerável prejuízo para a empresa. Processos de alta disponibilidade como os encontrados em negócios de Óleo & Gás, Açúcar & Etanol, Usinas Hidrelétricas, Usina Nuclear, Agroindústria, entre outros, demandam a utilização de sistemas redundantes para evitar que a planta pare de funcionar caso um equipamento fique indisponível.

A utilização desse tipo de arquitetura praticamente elimina a necessidade de parada do sistema para manutenção, ampliando a segurança operacional, aumentando a confiabilidade do sistema de automação e otimizando os custos e o desempenho do negócio. No trabalho de Liu et al. (2013) é apresentado uma forma de se fazer redundância em processos que não podem ser interrompidos, onde a redundância do controlador é feita por dois *CLPs* sincronizados.

Redundância, na automação industrial, significa manter sistemas duplicados ou triplicados para garantir a disponibilidade de processos e dispositivos críticos, sendo assim no trabalho Li, Xue e Gai (2014) é apresentado um sistema específico para fazer a votação entre os dados dos dispositivos de um sistema de controle de processos que possui redundância tripla. Existem

Figura 4 – SOA Aplicada na Automação de um Processo.



Fonte: Colombo et al. (2017).

diversos motivos para se utilizar sistemas de arquitetura redundante, mas é senso comum de que robustez, disponibilidade e segurança formam o tripé deste tipo de sistema:

- Robustez - capacidade de tolerar falhas e continuar operando o processo;
- Disponibilidade - como a própria denominação deixa claro, trata-se de todo o potencial do sistema estar disponível sempre que necessário;
- Segurança - mais do que apenas proteção de dados, a palavra segurança aqui se dá no contexto da segurança funcional, que resguarda a integridade física tanto dos operadores do processo quanto do patrimônio da empresa.

A utilização de redundância de hardware (ou de equipamentos) é um dos recursos mais empregados quando tolerância a falhas é requerida. Maior tolerância a falhas corresponde a maior disponibilidade da planta e maior segurança operacional e ambos os aspectos são importantes. A disponibilidade está diretamente relacionada ao tempo em operação e à lucratividade do negócio. E a segurança operacional diz respeito à preservação dos ativos e da vida das pessoas próximas ao processo.

Entre os tipos de redundância mais utilizadas na indústria está o hot stand-by, apresentado em Zhao e Liu (2004), técnica em que um ou mais módulos ficam em modo de espera enquanto o equipamento principal está operacional. Neste tipo de arquitetura, os módulos em espera funcionam em sincronia com o equipamento ativo e, caso uma falha seja detectada, o mesmo está pronto para se tornar operacional imediatamente. Para que não haja nenhum tipo de perda

entre a troca de equipamentos, o módulo em stand-by precisa estar sincronizado e atualizado com as mesmas configurações do equipamento principal.

1.1 Objetivo

O trabalho tem como objetivo o desenvolvimento e avaliação de uma nova abordagem para fazer o controle de processos de forma redundante como um serviço. O controlador redundante PID4.0 é baseado em uma modificação do algoritmo PIDPlus, para controle de processos em rede, para operação como um microsserviço.

1.2 Estrutura do Trabalho

Este trabalho de mestrado possui mais quatro capítulos, além deste capítulo introdutório. No Capítulo 2 é apresentada uma revisão da literatura sobre a revolução industrial e aplicações relacionadas à Indústria 4.0. É apresentado também as arquiteturas SOA, MOA e composição de serviços com orquestração, além de aspectos relacionados a virtualização de controladores e redundância em arquiteturas baseadas em serviços.

O Capítulo 3 apresenta o Material e Métodos, com a descrição das principais ferramentas utilizadas para a criação de microsserviços na plataforma *Node.js*. Abordaremos seus principais pontos como: o que é um serviço; finalidade do transportador de mensagens; Serviço *api gateway* padrão; Ações, métodos e eventos de um microsserviço. Além de uma breve descrição da Planta Piloto 4.0 utilizada, dos serviços e o desenvolvimento do serviço de controle PID4.0.

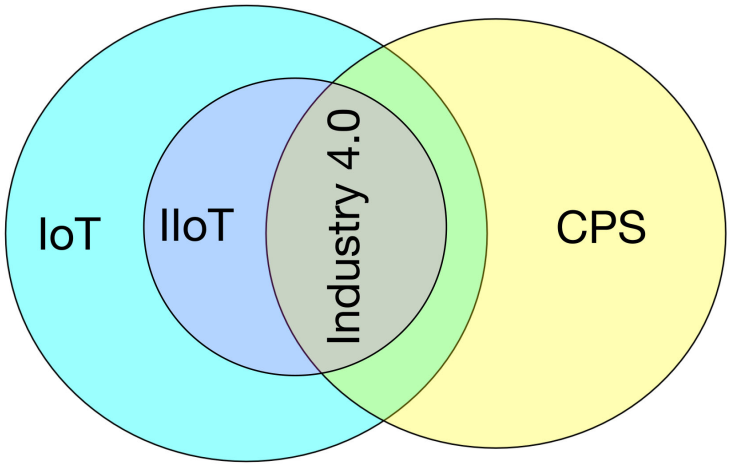
No Capítulo 4 é descrita a proposta desse trabalho, que tem como foco a realização de testes com o serviço de controle redundante PID4.0 para sua validação. A base de dados e monitoramento de processo é detalhado, além de como fazer a replicação dos serviços. Por fim no Capítulo 5 apresentamos as conclusões e por último estão as referências bibliográficas.

2 REVISÃO DA LITERATURA

2.1 Indústria 4.0

Surgiu na Alemanha em 2011 o conceito de Indústria 4.0 (JAZDI, 2014), ganhando visibilidade globalmente, utilizando de tecnologias de Internet para aumentar a eficiência de produção com serviços inteligentes (*smart services*) em fábricas inteligentes (*smart factories*). A I4.0 é um novo conceito que representa uma evolução dos sistemas produtivos atuais a partir da convergência entre novas tecnologias de automação e tecnologia da informação (BLANCHET et al., 2014). Levando em consideração a aplicação, podemos dizer que temos uma mistura entre a ideia de IoT com a ideia de Sistemas Cyber-Físicos (CPS) (SISINNI et al., 2018).

Figura 5 – Representação das intersecções entre IoT, IIoT, CPS e Indústria 4.0



Fonte: Sisini et al. (2018).

Alguns princípios fundamentais são recorrentes nas diversas definições de Indústria 4.0 introduzidas ao longo do tempo (BASSI, 2017). São eles:

- Uso extensivo da Internet: o uso de serviços de comunicação cabeada ou sem fio nos dispositivos inteligentes possibilita acesso direto às camadas superiores de processos e serviços. Isto eleva o valor agregado, promove suporte para o uso otimizado de recursos e o controle inteligente do processo de fabricação (JAZDI, 2014). Uma das principais vertentes é o uso de big data para a predição de falhas, de forma que se faça a manutenção antes que ocorra a quebra do equipamento. Isto reduz drasticamente o tempo de parada da planta e os prejuízos decorrentes da interrupção da linha (BASSI, 2017);
- Flexibilidade: a capacidade de operar linhas produtivas flexíveis, com capacidade de customização e minimização do tempo de reprogramação (setup), de forma a atender as

necessidades individuais do consumidor sem custo adicional por parada e reprogramação da linha de produção (JAZDI, 2014). Algumas tecnologias habilitadoras típicas são a impressão 3D e a rastreabilidade e identificação do produto na linha de produção (BASSI, 2017);

- Comunicação, Virtualização e CPS: a criação de modelos unificados em contraposição aos diversos protocolos industriais atuais (AS-i, PROFIBUS, PROFINET, CAN, etc.) que em geral são incompatíveis entre si ou de difícil interoperabilidade. A disponibilização de um framework comum de comunicação permite o uso de virtualização, conectando os sistemas físicos a contrapartes virtuais (BASSI, 2017).

Segundo Kagermann, Wahlster e Helbig (2013), o potencial da I4.0 fica evidente devido à algumas características e possibilidades, como:

- Atender aos requisitos individuais do cliente: A linha de produção pode se modificar para atender critérios individuais, específicos do cliente. Mudanças de última hora e volumes de produção muito baixos (tamanho de lote de 1), também são possíveis;

- Flexibilidade: É possível configurar dinamicamente diferentes aspectos dos processos de negócios, tais como qualidade, tempo, risco e preço. Isso possibilita a otimização do uso de materiais, os processos de engenharia podem ser tornados mais ágeis, processos de fabricação podem ser alterados, a escassez temporária de algum material (devido a problemas de fornecimento, por exemplo) pode ser compensada e grandes aumentos no tamanho dos lotes produzidos podem ser alcançados em um curto espaço de tempo;

- Tomada de decisão otimizada: A I4.0 permite a verificação antecipada das decisões de projeto, respostas mais flexíveis à interrupção e otimização global em todos os setores de uma empresa;

- Produtividade e eficiência dos recursos: A mesma motivação que resultou nas revoluções industriais anteriores impulsiona a I4.0, a obtenção da maior produção possível a partir de um determinado volume de recursos (produtividade dos recursos) e utilização da menor quantidade possível de recursos para produzir uma determinada quantidade de produtos (eficiência de recursos). Os processos de fabricação podem ser otimizados para cada caso individual, até mesmo sem parar a produção, sendo continuamente otimizados em termos de consumo de recursos e energia ou de redução de suas emissões.

As aplicações da I4.0 são baseadas na ideia de que tudo estará conectado para que as melhores decisões de produção, custo e segurança sejam tomadas, tudo sob demanda e em tempo real. Para que isso aconteça, é necessário a interconexão de dados de automação e infraestrutura de redes de TI de forma segura e confiável (WOLLSCHLAEGER; SAUTER; JASPERNEITE, 2017). As

arquiteturas industriais de automação e controle tradicionais não fornecem, principalmente, estes requisitos de interoperabilidade e integração vertical e horizontal de equipamentos e sistemas (BORANGIU et al., 2019) e (SISINNI et al., 2018). Em consequência, torna-se necessário o desenvolvimento de novas arquiteturas industriais que forneçam tais requisitos e suportem as demandas tecnológicas relacionadas à I4.0. Nesse sentido a utilização da SOA surge como uma solução para a integração das tecnologias já existentes para a I4.0.

2.2 Arquitetura Orientada a Serviços (Microsserviços)

A Arquitetura Orientada a Serviços (SOA - Service Oriented Architecture) é uma arquitetura de sistemas de Tecnologia da Informação na qual se busca fragmentar as aplicações convencionais em "Serviços", que são funções e processos individuais de negócio (XIAO; WIJEGUNARATNE; QIANG, 2016). Na prática, trata-se de uma arquitetura modular, na qual os elementos devem ser compostos e decompostos de acordo com as diferentes necessidades do negócio. Neste tipo de sistema, cada serviço deve ser passível de descoberta e independente quanto à plataforma e à linguagem (THEORIN; HAGSUND; JOHNSSON, 2014), com o objetivo de obter baixo acoplamento de um serviço em relação ao outro, escalabilidade e a capacidade de manter uma arquitetura distribuída.

SOA é uma nova tecnologia, que tem como filosofia de projeto a modularidade, separação de tarefas, reuso de serviços e composição. Também podendo ser vista como uma nova metodologia de programação baseada em padrões e ferramentas que envolvem o uso de serviços *web* em larga escala (ORDANINI; PASINI, 2008).

No contexto computacional, os serviços são módulos de negócio que possuem interfaces de conexão que são invocadas via mensagens (DUSTDAR; PAPAZOGLOU, 2008). Essas interfaces disponibilizam recursos sem que a implementação dos serviços seja conhecida, sendo esses serviços processos independentes que são descritos, disponibilizados, localizados e invocados por meio de redes de comunicação. Os serviços podem interagir através de redes de comunicação com simples respostas a requisições de processos até a execução de ações e tarefas complexas que exigem uma comunicação entre provedores e consumidores de serviços virtuais.

Existem diversas pesquisas focando no desenvolvimento de SOA, tendo como objetivo oferecer uma arquitetura que suporte propagação e a utilização de serviços virtuais reutilizáveis (Xiao, Wijegunaratne e Qiang, 2016; Jammes et al., 2014). Para Sheng et al. (2014), SOA é uma arquitetura que organiza infraestruturas e aplicações em um conjunto de serviços passíveis de interação, onde as informações e os recursos ficam disponíveis para todos os clientes como serviços independentes que são acessados de um modo padrão.

Por fim Beltrán, Iglesias e Fernández (2016) mostram os principais benefícios de se utilizar o SOA na industria:

- Agilidade - A manufatura ganha capacidade de se adaptar rapidamente a mudanças na produção se moldando as necessidades do cliente;
- Autonomia - O sistema se torna colaborativo onde os dispositivos são estruturados de uma maneira autônoma e distribuída. Os componentes do sistema se tornam funcionalmente unidades autônomas, onde a dependência um do outro é muito baixa ou quase nula;
- Integração Vertical e Horizontal - A Orientação por serviços permite o desenvolvimento de um gerenciamento colaborativo da manufatura;
- Interoperabilidade - As novas gerações de plantas industriais serão mais colaborativas e necessitarão de maior comunicação e compartilhamento de informação do que o padrão atual
- Escalabilidade - Em grandes sistemas é essencial manter a escalabilidade, tanto vertical quanto horizontal.

2.2.1 Microsserviços

Microsserviços é uma evolução do SOA, com um estilo arquitetônico em que as aplicações são decompostas em serviços acoplados, oferecendo modularidade, tornando o desenvolvimento, teste e manutenção das aplicações mais fácil, com menor perda de tempo. Com poucas responsabilidades, pequenos e autônomos, os microsserviços podem trabalhar de forma independente ou em conjunto com outros serviços. O conceito de SOA foi apresentado em 2014, descrevendo uma interpretação mais concreta da SOA em (BUTZIN; GOLATOWSKI; TIMMERMANN, 2016).

Definido como uma entidade separada e autônoma, cada microsserviço pode ser implementado em diferentes máquinas, e mesmo assim trabalhar de forma coletiva, tendo a capacidade de executar seus processos de forma independente, caso tenha falha de um outro serviço (MOREIRA; BEDER, 2015). Através de boas práticas de desenvolvimento de *software* com a intenção de gerar sistemas altamente escaláveis e com mudanças rápidas dos modelos de negócio em nuvem, surgiu os microsserviços. Empresas que tiveram sucesso com este modelo são por exemplo Netflix, Amazon e Soundcloud (BUTZIN; GOLATOWSKI; TIMMERMANN, 2016).

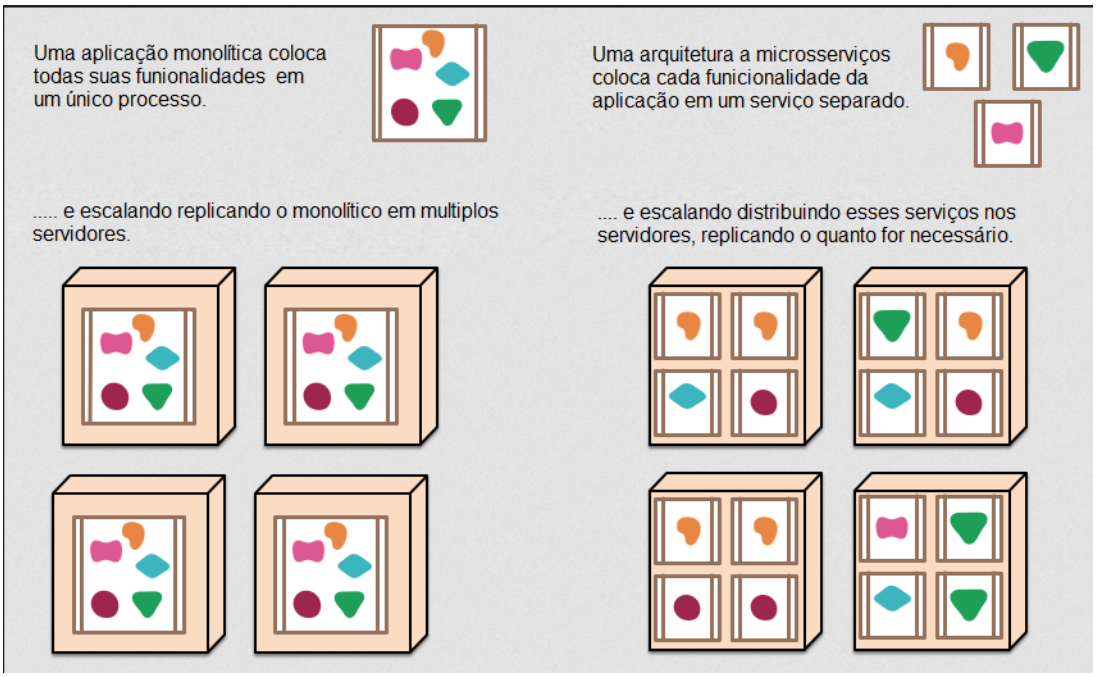
Fowler e Lewis (2014) e Newman (2021) definem as principais características de um microsserviço, que são:

- Cada microserviço implementa um único recurso de negócios ou funcionalidade;
- Um microserviço é o suficiente simples para ser desenvolvido e mantido por um único programador;

- Microserviços são executados em processos separados, comunicando-se por meio de padrões de mensagens ou APIs bem definidas;
- Microserviços não compartilham armazenamentos de dados. Cada microserviço é responsável por gerenciar seus próprios dados;
- Microserviços têm bases de código separadas e não compartilham código-fonte. No entanto, eles podem usar bibliotecas de utilitários comuns;
- Cada microserviço pode ser implantado e atualizado de forma independente de outros serviços.

Ao ser agnóstico de dispositivos e plataformas, os microsserviços fornecem flexibilidade para o sistema desenvolvido. Têm diferentes formas de comunicação e processamento sendo mais um dos atributos que os distinguem do SOA tradicional. A Figura 6 apresenta um esquemático dos microsserviços.

Figura 6 – Esquemático da composição dos microsserviços.

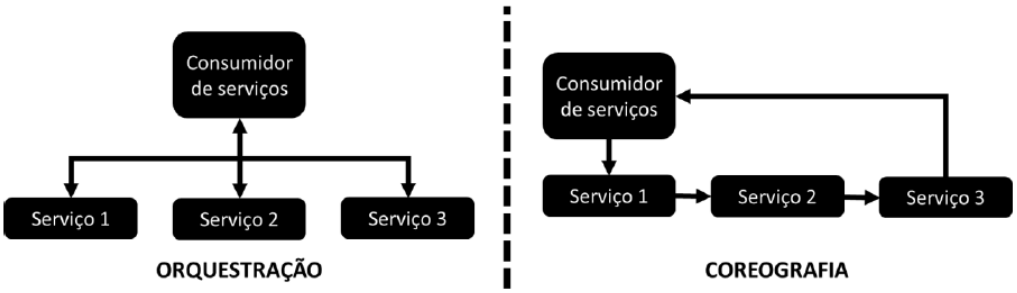


Fonte: Adaptado de Lewis e Fowler (2014).

2.2.2 Composição de Serviços

Quando tratamos de arquiteturas baseadas em serviços, devemos levar em consideração a organização que será adotada para que todo o conjunto funcione corretamente. Richards (2016) apresenta dois conceitos de composição de serviços, a coreografia e a orquestração. A Figura 7 apresenta um esquema comparativo entre orquestração e coreografia de serviços.

Figura 7 – Comparativo entre orquestração e coreografia de serviços



Fonte: Adaptado de Richards (2016).

A orquestração representa a composição de serviços para criar um novo serviço ou para resolver uma tarefa de um processo de negócio. Assim sendo, sempre existe um processo central ou mestre, sendo que este é responsável por coordenar o serviço ou uma atividade de negócio para chamar os demais serviços afim de compor uma função de maior relevância. Neste tipo de composição, cada serviço participante não tem conhecimento do processo como um todo e que faz parte de uma composição de serviço de maior hierarquia. Apenas o processo mestre detém a inteligência sobre toda a aplicação, em outras palavras, a orquestração de serviços se refere à coordenação centralizada de diversos serviços através de um mediador, sendo que este tem a responsabilidade de coordenar a execução dos serviços, obedecendo uma ordem pré definida para a execução.

Na coreografia, os serviços são pré-determinados antes da sua execução. Neste tipo de composição quando um serviço é acionado e envia uma mensagem, outros serviços podem estar programados antecipadamente para receber ou não essa mensagem e disparar outras ações. Sendo assim neste processo não há um sistema central ou mestre que coordena todo o processo. Nessa configuração, cada serviço envolvido tem o conhecimento de que faz parte de uma composição de serviços e que precisa interagir com outros serviços/aplicações de maneira ordenada para que a composição resultante tenha sucesso. Cada serviço sabe quando atuar, com quais outros serviços interagir, quais operações deve executar e quais mensagens deve trocar, sendo a execução, portanto, descentralizada.

A coreografia de serviços pode ser definida como um sistema que dispensa um elemento central coordenador, pois partimos da premissa que cada serviço sabe qual é sua parte dentro do contexto, onde a aplicação final é formada pela soma dos serviços individuais (BUTZIN; GOLATOWSKI; TIMMERMANN, 2016). Neste tipo de implementação, cada serviço é responsável por acionar o seu sucessor, formando o que se chama “cadeia de serviços”, de forma semelhante a uma coreografia de dança, na qual cada integrante conhece a sua função dentro do contexto e a executa sem que haja um comandante central.

Do ponto de vista das arquiteturas SOA, ambos os modelos podem ser implementados. Na

prática, porém, predomina o uso de coreografia, na qual há maior flexibilidade na gestão dos serviços (BUTZIN; GOLATOWSKI; TIMMERMANN, 2016). Quando um novo serviço é adicionado dentro de um contexto de orquestração, a aplicação mediadora deve ser alterada de forma a incluir o novo serviço. Em um contexto de coreografia, por sua vez, basta que os serviços relacionados sejam alterados, de forma que o novo serviço seja acionado quando necessário.

Um dos pontos relevantes quanto à escolha do tipo de organização é a necessidade ou não de se verificar que todos os serviços foram executados com sucesso. Em serviços orquestrados, esta verificação é possível diretamente através da instância do mediador. Quando os serviços são coreografados, esta verificação só pode ser implementada através de um serviço de monitoramento dos demais (BUTZIN; GOLATOWSKI; TIMMERMANN, 2016).

2.3 Aplicações Industriais usando Serviços

Ollinger et al. (2014) destacam um conceito de SOA utilizando um CLP permitindo uma geração totalmente automatizada e a implementação de funções de dispositivos de campo como serviços implementados utilizando o Devices Profile for Web Services (DPWS). Utilizando um CLP convencional da Phoenix Contact e a ferramenta de orquestração DPWS *JGrafchart* , é demonstrado através de um estudo de caso que supera as limitações atuais dos sistemas CLP existentes, atendendo às demandas de um ambiente de produção em rápida mutação e seus requisitos desafiadores de flexibilidade e reusabilidade.

Com este conceito, os blocos de funções são agrupados e encapsulados e, no campo de serviços da *Web*, disponibilizando fora do CLP as funções do dispositivo, permitindo um nível mais alto da pirâmide de automação. Consequentemente, a lógica de controle de um SOA-PLC não é mais realizada por um código sequencial inflexível, mas por uma orquestração dinâmica de serviços. A orquestração pode ser feita por qualquer ferramenta de orquestração DPWS como o *JGrafchart* em qualquer PC convencional ou até mesmo em outro CLP.

Beltrán, Iglesias e Fernndez (2017) demonstram a utilização do padrão REST para o desenvolvimento de uma arquitetura SOA industrial. No seu nível inferior, a arquitetura permite a comunicação com dispositivos e equipamentos ligados através de protocolos de comunicação tradicionais, como o Modbus TCP/IP ou através de conexão direta via GPIO. No nível mais alto da arquitetura, serviços são disponibilizados através de comunicação via protocolos compatíveis com o padrão REST.

Um caso de estudo de migração de uma planta de grande porte com diferentes equipamentos e rede de comunicação é apresentado por Carlsson et al. (2018), discutindo os desafios de uma migração de um sistema de controle de um processo tradicional para um SOA. Foi feito a troca dos sistemas de controle baseados em Controlador Lógico Programável para o SOA. Uma visão importante destacada no trabalho foi o desenvolvimento de um serviço de mediação para manter a compatibilidade da SOA com as soluções de rede do sistema legado. Esse serviço faz a

conexão/tradução entre as redes instaladas (Profibus-DP, Modbus TCP/IP e OPC UA) e o padrão de comunicação baseado em serviços *WEB* da SOA.

Bigheti (2020) e Fernandes et al. (2021) apresentam uma Arquitetura Orientada a Microsserviços, debatendo as características, detalhes operacionais, vantagens e potencial de aplicação a ser explorado. Os microsserviços são uma variante do SOA, porém com um formato arquitetônico no qual as aplicações são decompostas em serviços, permitindo modularidade o que torna o desenvolvimento, teste e implantação das aplicações mais fáceis. Por fim, estes trabalhos apresentam testes operacionais validando a utilização do MOA.

2.4 Virtualização de Controle e Redundância

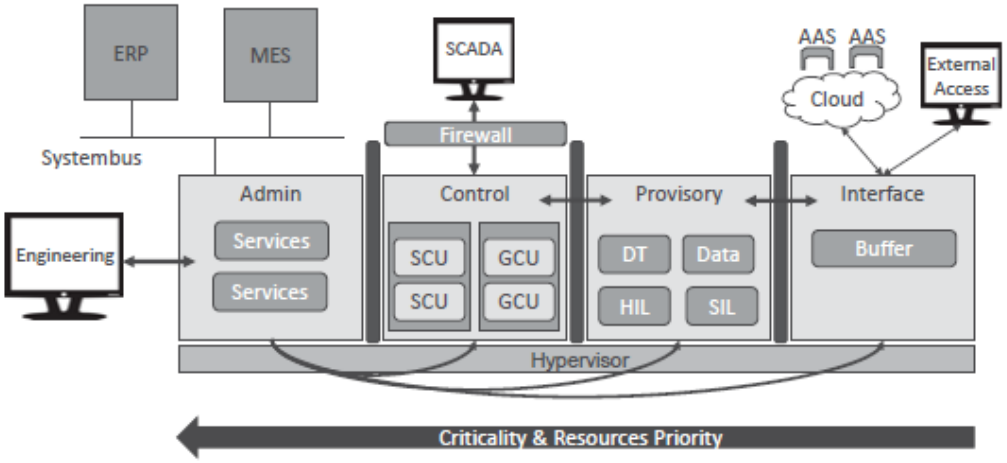
A automação em nuvem vêm se tornando uma realidade com o avanço de tecnologias de segurança da informação. Em Delsing (2017) é apresentado a evolução da indústria 4.0 com a digitalização dos meios de produção. Neste trabalho foi desenvolvido um sistema onde temos redes locais independentes, cada um com seu sistema de autorização, registro, orquestração, aplicações e suporte a automação o que facilita a virtualização.

Em Mellado e Núñez (2022) é proposto a virtualização de um CLP chamado de "IoT-PLC", que tem como principal foco a adaptação deste dispositivo a indústria 4.0 e este projeto busca atender esse novo conceito de se fazer controle, trazendo maior flexibilidade ao dispositivo de controle, permitindo multi-tarefas além do controle. O IoT-PLC é dividido em pequenas partes em forma de contêiner, onde cada pacote de funcionalidade é colocado em contêineres separados, permitindo capacidades de controle, funcionalidades de computação em rede como filtro e armazenamento de dados, além de um gerenciamento independente de interfaces wireless. Interessante mencionar que é utilizado como *hardware* a Raspberry Pi e que tem capacidade de interagir de forma transparente com níveis hierárquicos superiores da camada de rede.

A virtualização pode fornecer uma camada de abstração para o *software* aumentando a agilidade do sistema, além de fornecer um controle melhor dos recursos distribuídos em diferentes aplicações/sistemas operacionais. Baseado nisso em Azarmipour et al. (2019) é apresentado uma virtualização de um CLP, chamado de PLC 4.0, utilizando Hypervisor foi desenvolvido máquinas virtuais e proposto uma arquitetura para este sistema mostrado na Figura 8. Essa arquitetura permite uma isolação entre o físico e o virtual, permitindo uma execução independente.

Por fim Givehchi et al. (2014) apresentam uma virtualização de CLP, já que cada CLP novo instalado demanda novos custos e adaptações no processo, e a grande questão do trabalho é responder se é possível transformar a função de um controlador em um serviço e este ter sua performance comparável a um CLP com hardware dedicado. Outra importante contribuição foi o trabalho (FERNANDES et al., 2020) que apresentam o CLP como um serviço, utilizando a arquitetura orientada a microsserviços.

Figura 8 – PLC 4.0.



Fonte: Azarmipour et al. (2019).

Redundância em sistemas na prática é manter os sistemas e processos operando de forma contínua e ou reduzindo ao máximo o tempo de parada desse sistema, a forma mais comum consiste em manter um sistema funcionando em paralelo monitorando o sistema que está ativo, em caso de falha desse o outro entra em operação sendo capaz de executar as mesmas funções do original com o mesmo nível de confiabilidade.

Existem processos e sistemas que a falha de um componente pode ser catastrófico, nestes casos a única forma de manter o processo de forma segura e confiável é não ficar dependente de um único sistema de controle ou em caso de processo industrial uma única malha de controle e ou segurança. Uma forma de evitar isso é fornecer opções para que o processo se mantenha operacional em caso de falha.

A redundância de controladores usando tecnologias como a comunicação em rede e a computação em nuvem tem sido investigada recententemente. (HEGAZY; HEFFEDA, 2015), apresentam uma proposta de redundância em nuvem usando controladores redundantes alocados em diferentes localidades, com um compensador de possíveis atrasos inerentes à rede. Para ter confiabilidade no sistema de controle com redundância em nuvem, foi proposto um algoritmo de tolerância a falha que roda de forma assíncrona nos controladores redundantes, da seguinte forma:

- Se o primeiro controlador falhar, o controlador secundário estará automaticamente pronto para assumir. Essa é uma importante garantia para controlador redundante. Automaticamente se temos uma redundância tripla e o primeiro e segundo falharem, automaticamente o terceiro irá assumir o controle;
- Se o primeiro controlador estiver disponível novamente este irá voltar a assumir o controle, forçando o controlador secundário sair do controle;

Uma outra abordagem também levando em consideração o atraso da rede como um limitante para a virtualização de controladores apresentado em (MUBEEN et al., 2017), onde é defendido que para termos os grandes benefícios do IoT como escalabilidade, flexibilidade e eficácia de custo é necessário implementar uma rede local para proteger o sistema de automação de atrasos imprevisíveis encontrados na WAN. Neste cenário onde o controlador é alocado na rede local, a comunicação deste com os dispositivos pode ser feito em tempo real, transformando o controlador em um serviço.

Por fim o trabalho Bigheti, Fernandes e Godoy (2019) discute as funcionalidades e benefícios de se utilizar a arquitetura orientada a microsserviços na indústria 4.0. Para isso é apresentado alguns microsserviços utilizados no experimento apresentado como o DAQ e o controle PIDPlus. O PID4.0 apresentado neste trabalho é uma evolução do algoritmo de controle do PIDPlus que permite a redundância deste tipo de controlador.

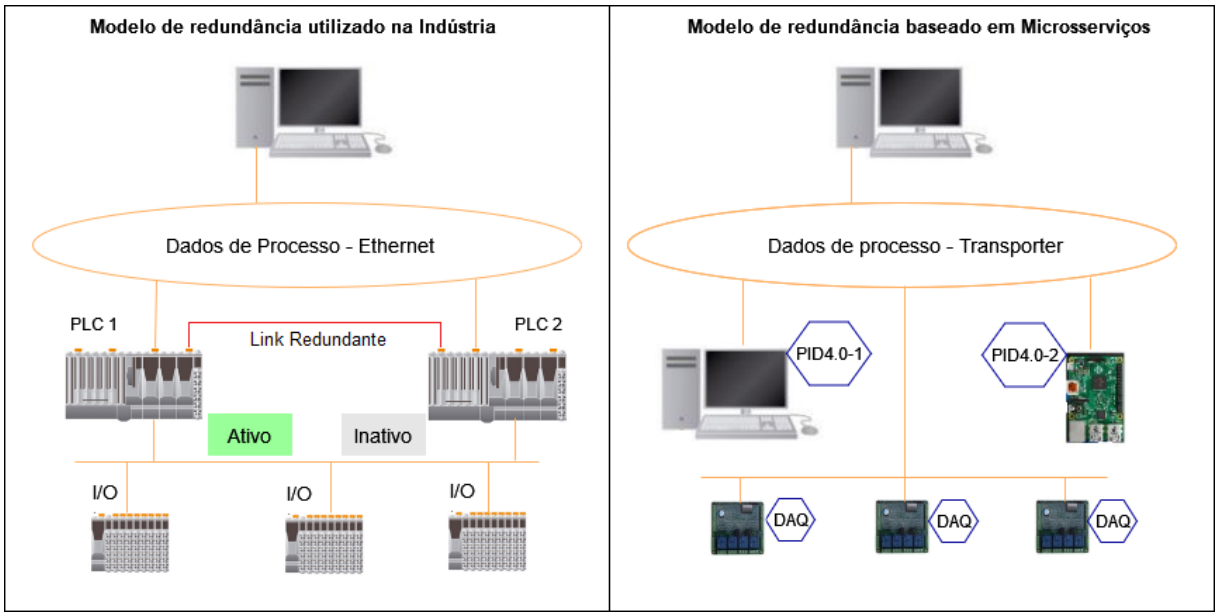
3 MATERIAIS E MÉTODOS

3.1 Proposta do Trabalho

Na indústria de processos as aplicações de redundância podem ser de diversas formas, porém neste trabalho o foco é na redundância de controladores. Geralmente tem-se vários controladores redundantes (em paralelo) controlando um mesmo processo, a redundância de controladores acontece com a replicação de toda uma estrutura de *hardware/software* complexa. Se torna mais complexo devido a algoritmos de programação adicionais ou à transferência do controle dos componentes originais para os redundantes, além de necessitar de um link redundante interconectado aumentando a complexidade do *hardware* do sistema.

Neste trabalho essa redundância de controladores será feito utilizando o formato de micros-serviços, não necessitando de *hardware* dedicado e diminuindo a complexidade do sistema, já que não necessita do link redundante e nem de programações adicionais conforme Figura 9. A proposta deste trabalho visa o desenvolvimento de um controlador redundante que chamaremos de PID4.0, e para isso será utilizado a Planta Piloto 4.0 para seu desenvolvimento e testes.

Figura 9 – Modelo de controlador redundante utilizado na indústria e baseado em microsserviços.

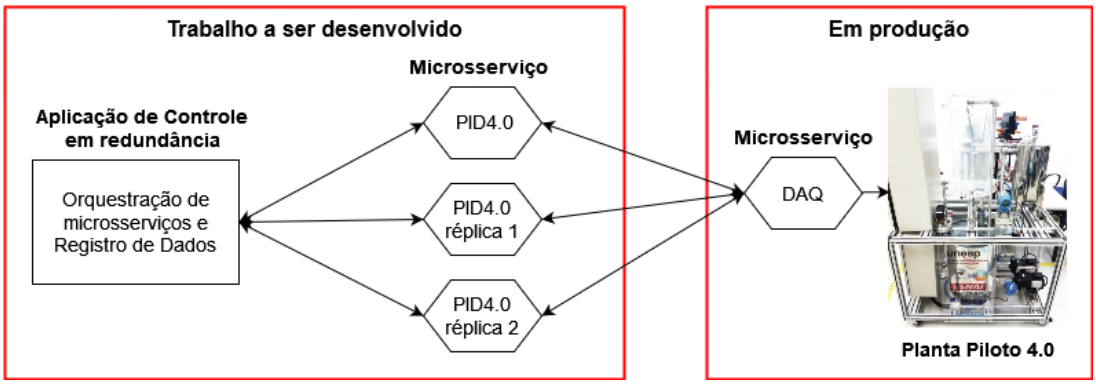


Fonte: Autor.

Esse projeto propõe o desenvolvimento e testes experimentais com microsserviços de controle redundante com foco na indústria 4.0. Para isso será utilizado a tecnologia de serviços baseada no *framework* Moleculer de microsserviços. Na Figura 10 é apresentado um esquemático com os processos a serem desenvolvidos. Conforme mostrado na Figura 10 o primeiro processo é o desenvolvimento de uma aplicação que faça o controle redundante (orquestração) e também

armazene os dados em um registrador, que serão utilizados pelos microsserviços de controle PID4.0. O microsserviço PID4.0 será desenvolvido de forma que possa ser alocado suas réplicas em diferentes instâncias permitindo a redundância. Para realizar este desenvolvimento e testes será utilizado a planta piloto 4.0 e o Microsserviço DAQ já desenvolvido no trabalho (PONTAROLLI, 2020).

Figura 10 – Diagrama do PID4.0: Controlador Redundante de Processos como um Serviço.



Fonte: Autor.

3.2 Framework Moleculer

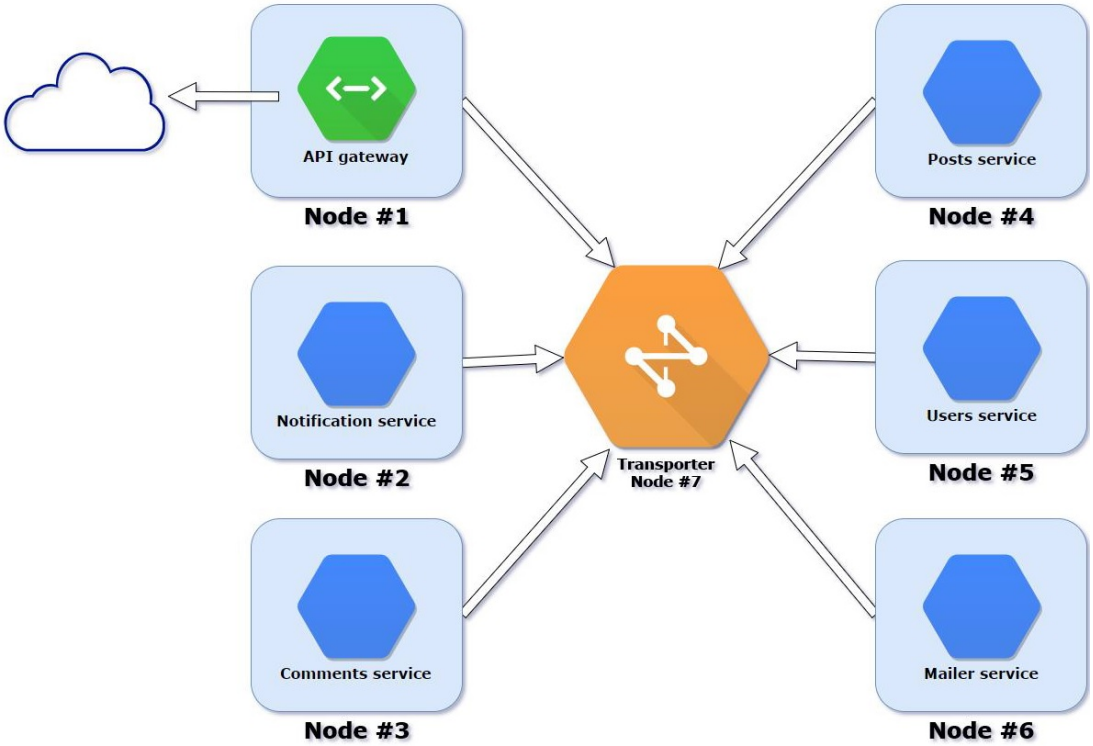
Um *framework* tem uma estrutura base ou uma plataforma de desenvolvimento, que contém bibliotecas, ferramentas, sistemas e componentes que simplificam o processo de desenvolvimento de uma solução específica. O Moleculer é um *framework* de código aberto utilizado para criar microsserviços em linguagem JavaScript para a plataforma Node.js. Ele contribui na construção de serviços eficientes, confiáveis e escaláveis, fornecendo muitos recursos para construir e gerenciar microsserviços (MOLECULER, 2021). A utilização deste *framework* está bem detalhada em (PONTAROLLI, 2020), aqui apresentaremos um breve resumo.

A Figura 11 apresenta a arquitetura orientada a microsserviços (MOA) do *framework* Moleculer, na qual os serviços são executados em nós individuais que se comunicam via protocolo de comunicação (*Transporter*) através de um *broker*, ou seja, um intermediário de envio de mensagens. Nesta arquitetura, a latência da comunicação entre os serviços não é insignificante, mas os serviços podem ser replicados para proporcionar resiliência e tolerância a falhas. Outras estruturas de interconexão entre os serviços também são permitidas no Moleculer, permitindo a melhor configuração para cada aplicação.

A replicação de serviços é a implementação de dois (ou mais) serviços idênticos com mesmo nome, em instâncias de execução diferentes. Na prática, o Moleculer pode fazer as requisições para qualquer um dos serviços replicados, dependendo da disponibilidade de cada um dos serviços. Tendo também a possibilidade de reuso dos códigos, porém estabelecendo cada serviço

com um nome distinto, sendo que nesse caso, o Moleculer direciona as requisições para cada serviço especificamente.

Figura 11 – Arquitetura Orientada a Microserviços do Moleculer.



Fonte: Moleculer (2021).

Para um reuso de código mais eficiente, se temos por exemplo um serviço com a funcionalidade de controlar o nível de algum liquido em um tanque, e surge a necessidade de se implementar esse mesmo controle para outros tanques, por exemplo, o reuso pode ser feito de forma automática com algumas alterações do código a ser aplicado, tendo as mesmas funcionalidades para todos os controles, porem de maneira individual.

A arquitetura do Moleculer não tem qualquer tipo de hierarquia ou prioridade entre os microsserviços que compõem. Uma grande vantagem desse *framework* é que todos os microsserviços desenvolvidos possuem disponível um recurso automático de registro e descoberta, sendo assim, todos os serviços existentes são informados após a criação de um novo serviço ou após ser disponibilizado uma nova funcionalidade em um serviço. Um outro recurso relevante é o balanceamento automático de carga, que tem a função de distribuir dinamicamente a carga da comunicação entre os microsserviços uniformemente.

Os quadrados de cor azul na arquitetura (Figura 11) representam os nós que fazem o gerenciamento do Moleculer para dar suporte ao desenvolvimento, descoberta e configuração dos (micro) serviços. Cada Nó é responsável por gerenciar seus dados, possuindo seu próprio banco de dados, podendo conter um ou mais serviços. Os serviços podem executar e oferecer diferentes tarefas

que são chamadas de ações (exemplo: um serviço que faz aquisição de dados pode fornecer uma ação de aquisição de dados de entrada e uma ação de atualização de dados de saída). Na Figura 11 cada serviço disponibilizado na arquitetura é representado com um hexágono azul.

O hexágono verde representa o (micro) serviço de *gateway* (API *Gateway*), que tem a função de interface de conexão dos serviços internos (hexágonos azuis) com as aplicações externas por meio de chamadas via rede ou nuvem através da arquitetura de *software* REST. A comunicação entre os microsserviços é feita via um serviço de mensagens (*Transporter*), conhecido também como *Broker* de mensagens, representado pelo hexágono laranja.

O serviço de mensagens (*Transporter*) é um dos principais componentes da arquitetura, pois é o meio que coordena o envio e a recepção de mensagens em uma fila. O *Transporter* é projetado para enfileirar e manter as mensagens até serem processadas por consumidores. O produtor simplesmente envia a mensagem para o *Transporter*, sem a necessidade de um mecanismo de descoberta de serviços.

A comunicação é baseada em mensagens, suportando uma variedade de padrões de comunicação, incluindo os pedidos de caminho único (*one-way requests*) e *publish-subscribe*. Um benefício desse serviço no Moleculer, é que se um mesmo serviço for disponibilizado em múltiplas instâncias, e em diferentes nós (maior disponibilidade), as requisições recebidas pelo *Transporter* serão automaticamente balanceadas entre esses nós que estão online naquele momento e que podem executar o serviço requisitado. Além disso, usando o *Transporter* a mudança de um mecanismo de comunicação para outro (entre os diferentes disponíveis no Moleculer) é transparente.

No Moleculer, o serviço de *Transporter* utiliza um protocolo de comunicação para se comunicar com os demais microsserviços. A grande vantagem dessa abordagem é o desacoplamento entre produtores e consumidores de dados e eventos. O desacoplamento simplifica o desenvolvimento aumentando a disponibilidade, em comparação ao uso de transações distribuídas. Caso não tenha consumidor disponível para processar um evento, o *Transporter* irá enfileirar o evento até que a mensagem possa ser processada.

O serviço *Transporter* pode estabelecer a comunicação entre sistemas completamente diferentes de uma maneira confiável. O *Broker* de mensagem permite a integração de sistemas de forma que eles possam trocar dados de maneira desacoplada. A troca de informações entre os microsserviços utilizando o *Transporter* é feita pelo módulo *Serializers*, responsável pela serialização dos dados das mensagens.

Os principais recursos para microsserviços que compõem o framework Moleculer são (MOLECULER, 2021):

- Conceito de requisição-resposta;
- Arquitetura baseada em eventos;

- Suporte a middlewares para Node.js;
- Suporte ao armazenamento em cache de variável;
- Diversas opções de comunicação (Transporters);
- Diversas opções de serializers: JSON, MsgPack, Protocol Buffer e etc;
- Suporte ao desenvolvimento de múltiplos serviços em um mesmo nó;
- Suporte nativo para o registro de serviços;
- Descoberta automática de serviços;
- API para interface com aplicações externas.

O *framework* Moleculer propicia o desenvolvimento de um estilo arquitetônico em que as aplicações são decompostas em serviços de baixo acoplamento, disponibilizando modularidade e tornando mais simples a sua implementação. As principais classes de componentes operacionais do *framework* Moleculer são:

- *ServiceBroker*;
- *Service*.

3.2.1 *Service Broker*

O *ServiceBroker* é o componente principal da estrutura do Moleculer, pois engloba as características do *framework*. Responsável por lidar com serviços, chamar ações, emitir eventos e se comunicar com nós remotos. Tendo como função também a configuração dos nós como: nome do nó, registro de serviços, descoberta automática de serviços, *cache* de variáveis, *serializer*, *middleware* e *transporter*.

Para instanciar um *Node* (nó), é necessário carregar o módulo Moleculer no Node.js. Feito isso, pode-se criar um novo objeto de *ServiceBroker* e na sequência configurar as operações de funcionamento do microsserviço. É necessário criar uma instância do *ServiceBroker* para cada nó (MOLECULER, 2021).

3.2.2 Serviço (*Service*)

O Serviço representa um microsserviço no *framework* Moleculer, sendo possível definir ações e disparar eventos. Para criar um serviço, devemos definir um esquema (*schema*) contendo algumas partes principais: nome, versão, configurações, ações, métodos e eventos (MOLECULER, 2021).

Nome (*name*): Nome do microsserviço. É a primeira parte do nome da ação quando se chama o microsserviço. O *name* é uma propriedade obrigatória na criação do microsserviço.

Versão (*version*): Essa propriedade é usada para executar várias versões do mesmo microsserviço com ações diferentes.

Configurações (*settings*): Essa propriedade tem a função de permitir a configuração dos recursos disponíveis nos microsserviços. Por exemplo, a configuração da porta de comunicação dos comandos que serão enviados por meio do microsserviço *Gateway*. Essas configurações são enviadas durante o procedimento de descoberta do serviço;

Ações (*actions*): As ações são os métodos possíveis de serem chamados no serviço. A chamada de uma ação representa uma chamada de procedimento remoto (RPC – *Remote Procedure Call*). Tem parâmetros de solicitação e retorna a resposta, como uma solicitação Hyper Text Transfer Protocol (HTTP) (MOLECULER, 2021).

O método broker.call é utilizado para chamar uma ação interna. Quando o método broker.call é acionado, o ServiceBroker procura o microsserviço em um Nó que tem uma determinada ação e assim a chama. A função retorna uma Promise, que em JavaScript, é um objeto usado para processamento assíncrono. Uma Promise (promessa) retrata um valor que pode estar disponível imediatamente ou no futuro. Isso permite a combinação de métodos de tratamento para eventos da ação assíncrona num caso eventual de sucesso ou de falha. Isto permite que métodos assíncronos possam retornar valores como métodos síncronos: ao invés de um valor final, o método assíncrono retorna uma promessa ao valor em algum momento no futuro.

Métodos (*methods*): Os métodos são funções privadas do microsserviço, que somente são executadas internamente. As funções privadas, não podem ser chamadas através do *broker.call*. Para executar chamadas internas dos manipuladores de ações e de eventos, é necessário que se faça através da função *methods*.

Eventos (*events*): O *ServiceBroker* tem um barramento integrado para suportar uma arquitetura orientada a eventos. Quando acontecer um evento, uma mensagem pode ser enviada para os nós locais e remotos. A propriedade *events* (eventos) corresponde a de produtores de eventos que geram um fluxo de dados, e consumidores dos eventos que os escutam. Os eventos são entregues após a execução da propriedade, permitindo que os consumidores respondam imediatamente conforme a ocorrência desses eventos.

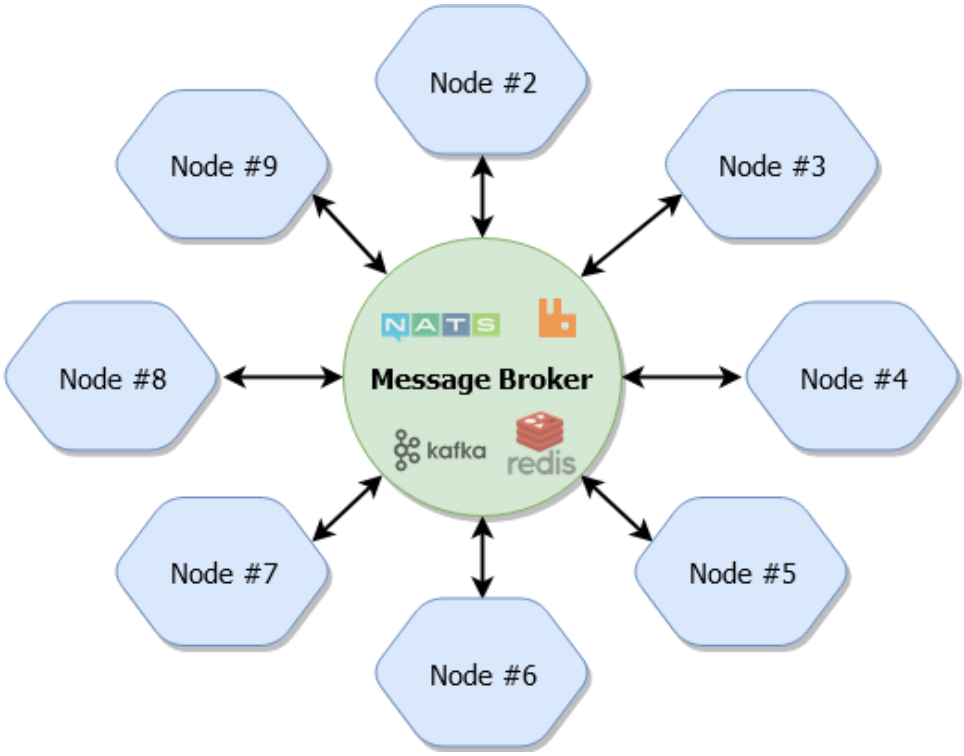
As arquiteturas de microsserviços baseados em eventos usam o modelo de *publish/subscribe*. Quando um evento é publicado, ele é enviado para cada um dos assinantes. Depois que um evento é recebido, não poderá ser reproduzido e não será exibido para assinantes novos. No Moleculer temos dois tipos de eventos que serão apresentados:

- *Balanced events*;
- *Broadcast event*.

3.2.3 Transporter

O serviço de mensagens (*Transporter*) tem a função de fazer a troca de mensagens entre os microsserviços da arquitetura MOA, o que facilita no desenvolvimento de uma aplicação, já que a comunicação é feita de forma automática e transparente conforme é apresentado na Figura 12. O *framework* Moleculer disponibiliza uma série de mecanismos diferentes para comunicação entre os microsserviços como: TCP/IP, NATS, MQTT, AMQP e etc.

Figura 12 – Microserviço *Transporter* da Arquitetura MOA.



Fonte: Moleculer (2021).

A configuração do tipo de *Transporter* é feita no método *ServiceBroker* do microsserviço na opção *Transporter*. Depois de ser configurado o tipo de *broker* de mensagem (*Transporter*), a comunicação entre os microsserviços ocorre de forma transparente, portanto, não importa o tipo de mecanismo de comunicação utilizado, não é preciso nenhum tipo de especificação ou configuração adicional para a comunicação entre os microsserviços.

As chamadas internas das ações serão executadas pela função *broker.call*. As chamadas externas das ações ocorrem através do microsserviço *Gateway*. O padrão de troca de mensagens utilizado é o método *publish/subscribe*.

3.2.4 *Gateway* (API)

O serviço *Gateway* (API) tem a função de fazer a interface padronizada de conexão dos serviços do Moleculer com aplicações externas. Para acessar o microsserviço *Gateway*, deve-se

usar o IP do dispositivo na porta 3000. Alguns exemplos de URLs da API *Gateway* para chamada de ações ou informações disponíveis nos microsserviços são descritos a seguir:

- Chamada da ação *hello* do microserviço teste: http://IP:3000/test/hello;
- Chamada da ação *add* com parâmetros do microsserviço *mat*: http://IP:3000/math/add?a=25&b=13;
- Chamada de informações do nó (microsserviço) *health*: http://IP:3000/ node/health;
- Chamada de listagem das ações de todos nós (microsserviços): http://IP:3000/ node/actions

3.2.5 *Load Balancing*

O balanceamento da rede, são configurações possíveis de estratégias para definir a política de requisição dos serviços existentes pelo *Transporter*, tendo como exemplo de configuração a Figura 13. Os tipos possíveis de configuração são *Round-Robin strategy*, *Random strategy*, *CPU usage-based strategy* e *Latency-based strategy*.

- *Round-Robin strategy* - Com essa estrategia o nó é selecionado através de um algoritmo round-robin que vai chamando os serviços de forma sequencial, conforme sequência que foram colocados on-line por exemplo $(1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 3)$;
- *Random strategy* - Utilizando essa estrategia o nó é selecionado de forma aleatória;
- *CPU usage-based strategy* - Utilizando essa estrategia o nó selecionado será o que tiver a menor utilização da CPU. Se tivermos uma grande quantidade de nós, essa estrategia irá fazer uma amostra e direcionar para o nó com menor uso de CPU, ao invés de fazer uma analise na CPU de todos os nós;
- *Latency-based strategy* - Com essa estrategia o nó que tiver a menor latência será o selecionado, que será medido através de comandos de *ping* feito periodicamente. Também se tivermos muitos nós será feito uma amostra e será selecionado o de menor latência.

3.3 Planta Piloto

A planta piloto industrial foi desenvolvida através de uma parceria entre a FAPESP, Unesp Sorocaba, SENAI de Lençóis Paulista e a empresa *Emerson Automation Solutions* e está bem detalhada em (PONTAROLLI et al., 2020) e (PONTAROLLI, 2020). A proposta da planta foi desenvolver e investigar o uso de arquiteturas orientadas a microsserviços (MOA) para aplicações de controle, automação e SCADA no contexto da Indústria 4.0. É possível ter uma visão geral dos componentes como atuadores e sensores na Figura 14, com uma legenda na Figura 15.

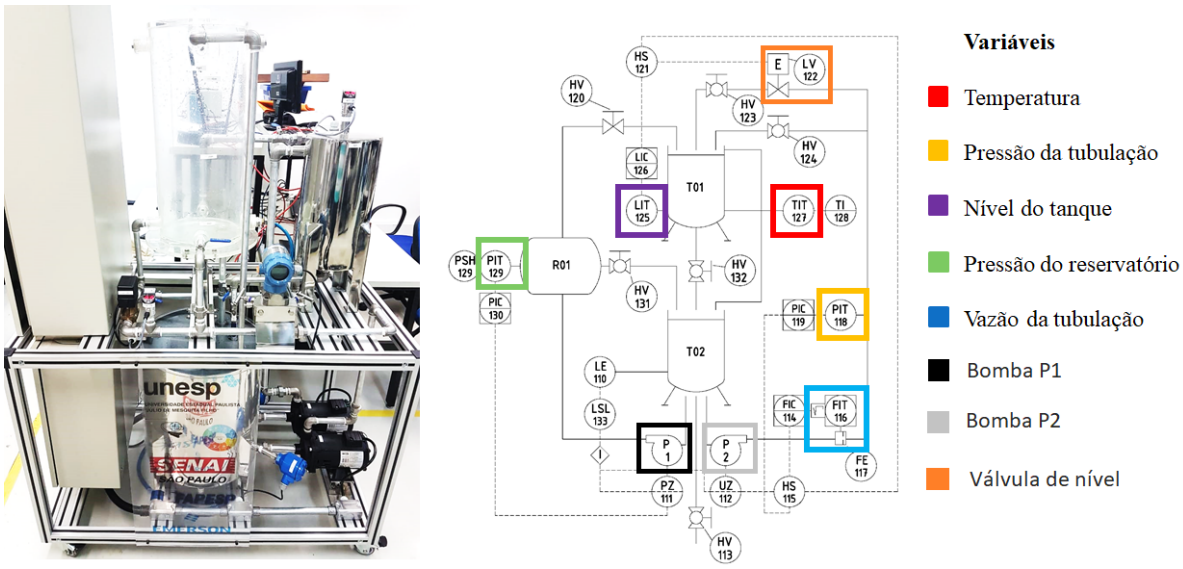
Figura 13 – Exemplo de configuração de Latência (*Latency-based strategy*).

```
// moleculer.config.js
module.exports = {
    registry: {
        strategy: "Latency",
        strategyOptions: {
            sampleCount: 15,
            lowLatency: 20,
            collectCount: 10,
            pingInterval: 15
        }
    }
};
```

Fonte: Moleculer (2021).

A planta piloto possui um tanque em inox principal (TQ02) de 83 litros, na parte inferior para armazenamento de água. O liquido pode ser bombeado pelos tubos de inox de ½" para parte superior da planta, com duas possibilidades. Uma rota para o tanque em acrílico (TQ01) de 38 litros utilizando a a bomba (P2), e uma outra rota para o reservatório de pressão em inox (R01) de 15 litros utilizando a bomba (P1).

Figura 14 – Planta piloto industrial.



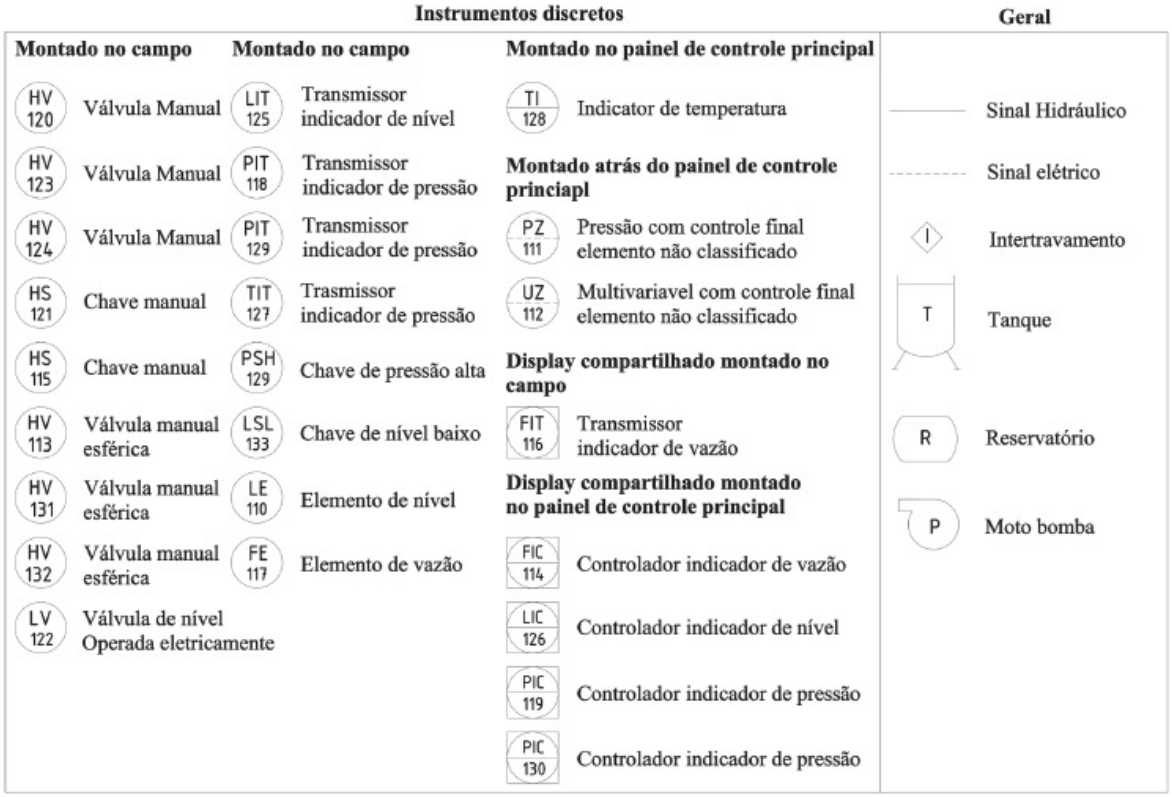
Fonte: Autor.

Os painéis elétricos foram divididos em dois gabinetes a fim de organização e facilidade de manutenção. O painel que fica na parte de baixo foi colocado os circuitos de potência e comando e na parte superior as interfaces de conexão de IO e placas programáveis. O painel superior possui também um display LCD touch de 10" para supervisão das informações da planta.

Temos as seguintes variáveis de processos, que podem ser medidas através dos instrumentos de medição da Emerson, que são: nível do tanque (LIT125), vazão (FIT116) com o transmissor de pressão coplanar de 0 a 5000 mmH2O da ROSEMOUNT modelo 2051CD, pressão de linha (PIT118) e pressão do reservatório (PIT129) com o pressostato eletrônico de 0 a 2,5 bar da WIKA modelo PSD-4, e temperatura (TIT127). Para medição de vazão é usado uma placa de orifício junto ao transmissor de pressão.

Como variáveis de comando, utilizamos duas bombas e uma válvula proporcional. As bombas são de baixo ruído do fabricante Dancor modelo Pratika CP-4R (P1 e P2), com as mesmas características de potência de 0,5CV e uma vazão de 4,24 m³/h e podem ser controladas por inversores de frequência da WEG modelo CFW300. Para evitar que as bombas operem sem água no reservatório incluímos uma chave de nível (LSL110) da SITRON, o que proporciona segurança na operação com as bombas. A válvula proporcional motorizada (LV 122) de ½" é da BUSCHJOST modelo 8288200.9650.

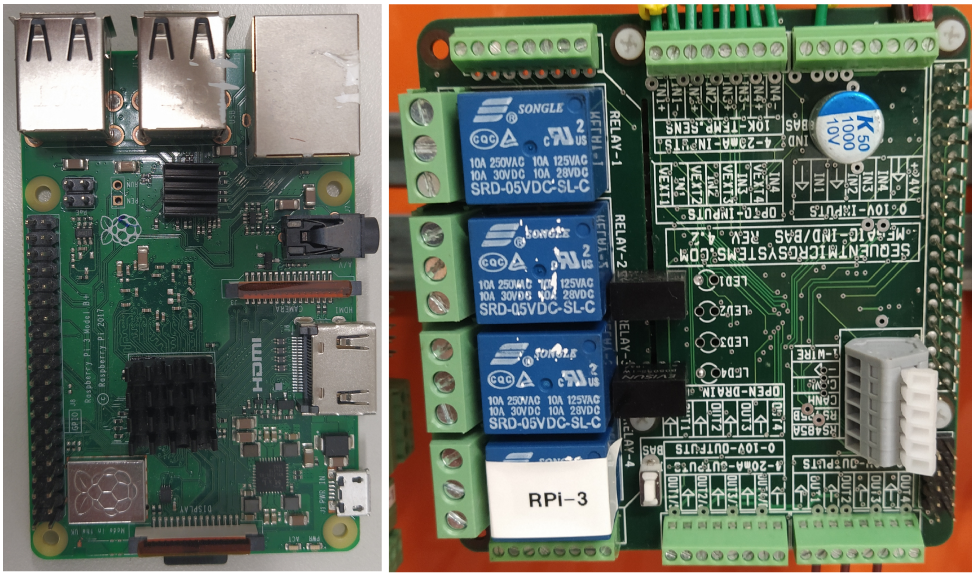
Figura 15 – Legenda do PI&D da planta piloto industrial.



Fonte: Autor.

No painel de controle é utilizado a placa *Raspberry Pi* 3B+, para a parte lógica de programação e microsserviços, e a placa de expansão MegaIO industrial da *Sequent Microsystems* (MegaIO, 2020), que será utilizada para aquisição de dados, e essa placa pode se conectar como um *shield* na *Raspberry Pi*, conforme Figura 16 .

Figura 16 – Raspberry Pi 3B+ e MegaIO Industrial.



Fonte: Autor.

Através da placa MegaIO, é possível fazer a coleta de corrente analógica disponibilizada pelos sensores na corrente de 4 a 20mA, bem como o acionamento das bombas gerando um sinal de tensão de 0 a 10V para os inversores de frequência, ou para a válvula proporcional. Este conjunto de placas *Raspberry Pi* com *MegaIO*, fazem parte do *hardware* responsável por fazer a aquisição dos dados e foi utilizado para o desenvolvimento do Microsserviço de Aquisição de Dados (DAQ) (PONTAROLLI, 2020).

Em resumo, com a utilização destes hardwares (*Raspberry Pi* e *MegaIO*) temos a capacidade de desenvolver um microsserviço que poderá fazer a leitura dos dados dos sensores e fazer a atualização das saídas dos atuadores, sendo uma aplicação fundamental para o controle dos processos da planta.

3.3.1 Serviços

Foram utilizados os sistemas operacionais Windows 10 no computador, contando com o software LabVIEW instalado para facilitar o controle, monitoramento e análise dos dados estatísticos. Na *Raspberry Pi* foi instalado o sistema operacional Linux, mais especificamente o sistema operacional oficial Raspbian, lembrando que a *Raspberry Pi* executará a parte lógica de todos os serviços. Esses serviços foram desenvolvidos utilizando o *framework* Moleculer. A estrutura é de código aberto e é feita na plataforma *Node.js* em conjunto com seu gerenciador de pacotes oficial NPM (Node Package Manager), tornando a criação de serviços mais simples, eficientes, confiáveis e escaláveis.

A ferrramenta *framework* Moleculer oferece diversas maneiras de fazermos a comunicação entre os microsserviços que são conectáveis e são do seguinte tipo: TCP (Transmission Control

Protocol)/IP, NATS, MQTT (*Message Queuing Telemetry Transport*), AMQP (*Advanced Message Queuing Protocol*) etc. Neste projeto foi utilizado o transporter NATS, que é o padrão do *framework* Moleculer que representa um sistema de mensagens com código aberto, simples e de alto desempenho para arquiteturas nativas de aplicativos, mensagens de microsserviços e IoT.

O tipo de transporter pode ser configurado no intermediário de serviço, podendo ser um dentre os disponíveis e que atenderá melhor o tipo de projeto desejado. Assim que o transporter estiver configurado, a comunicação entre os microsserviços ocorre de forma transparente, portanto, não importa a forma de comunicação que é utilizada, não sendo necessário nenhum tipo de especificação ou configuração adicional para a comunicação entre os microsserviços.

O serviço de transporter fica embarcado em uma *Raspberry Pi* utilizando uma rede *Wi-Fi* que é responsável pela troca de mensagens entre os microsserviços da arquitetura MOA, onde a comunicação entre eles é feita utilizando como meio o transporter NATS.

Para fazer chamadas internas das ações é utilizado o transporter, onde são executadas pelo intermediador do serviço de cada microsserviço, que irá disparar uma solicitação para qualquer um dos outros serviços que desejar que por sua vez também tem um intermediador de serviço próprio e distinto, que automaticamente responderá a chamada solicitada.

As aplicações externas são feitas através do serviço API *Gateway* que é responsável por essa interface padronizada de conexão entre os serviços do *Moleculer* e as aplicações externas, e para isso utiliza o protocolo REST. Possui alguns recursos dentre eles o *Hyper Text Transfer Protocol* (HTTP) e o *Hyper Text Transfer Protocol Secure* (HTTPS), múltiplas rotas, arquivos estáticos, suporte a autorização e analisadores de corpo JSON.

Para requisitar um microsserviço é necessário utilizar um navegador ou qualquer aplicação externa que tenha o padrão HTTP disponível, tendo isso é possível então colocar o endereço IP (*Internet Protocol*) de onde o microsserviço *Gateway* está embarcado e escutando na porta 3000 e se necessário passar um parâmetro, caso a ação deste serviço necessite, tendo como resultado um caminho como este: http://IP:3000/serviço/ação?parametro=valor.

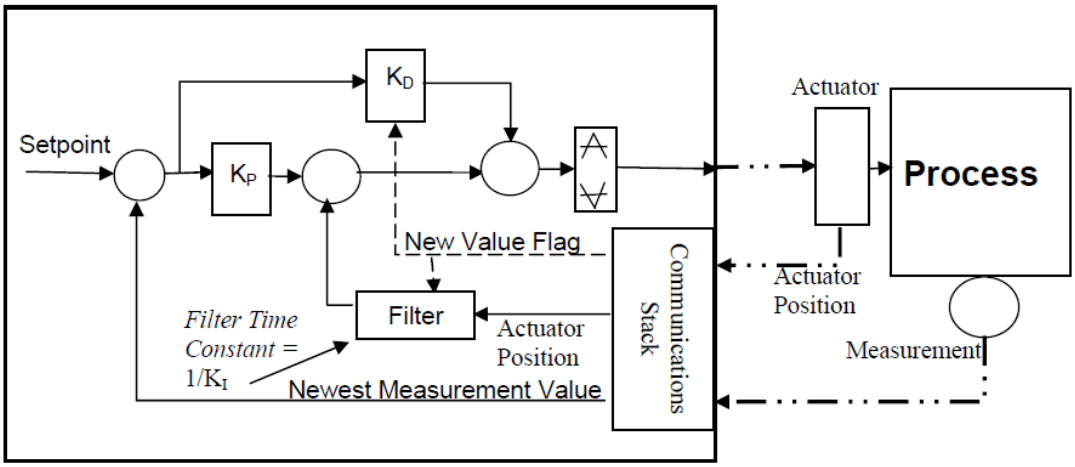
O microsserviço de Aquisição de dados (DAQ) tem como responsabilidade a aquisição de dados das variáveis de processo, utilizando módulos de *hardware* alocados no processo. Este serviço está alocado em uma *Raspberry PI* que se comunica com a placa *MegaIO* industrial através do protocolo I2C (*Inter Integrated Circuit*).

O microsserviço DAQ tem uma ação de leitura das entradas de corrente de 4 a 20 mA, *riin*, e uma outra *riin* para a leitura de dados de tensão de 0 a 10V, utilizadas para aquisição dos valores medidos pelos sensores da planta industrial que são: sensor de pressão de linha PIT118, sensor de pressão do tanque PIT129, sensor de temperatura TIT127, sensor de vazão FIT116 e sensor de nível do reservatório LIT125. Este serviço também conta com uma outra ação que é atualizar as saídas, *wuout*, com valores entre 0 e 10V, usados para atuar nos inversores de frequência das bombas P1 e P2 ou na válvula proporcional.

O Microsserviço controle *PIDPlus* foi desenvolvido e está operando com excelência na planta piloto 4.0 conforme previsto no trabalho (PONTAROLLI, 2020). Este controlador serviu como base para o desenvolvimento do microsserviço de controle com redundância (*PID4.0*). Para o desenvolvimento do *PID4.0* foi utilizado uma versão modificada do algoritmo *PIDPlus* (SONG et al., 2006). O Controle *PIDPlus* tem um algoritmo que foi criado para aplicações de controle em malha fechada via redes de comunicação, mas para trabalhar no conceito de requisições e operar como um serviço tivemos que modica-lo. Importante salientar que seria possível desenvolver outros microsserviços de controle com outros algoritmos e técnicas de controle diferentes, o que traria flexibilidade e modularidade no desenvolvimento e implementação.

O algoritmo tradicional do *PIDPlus* (SONG et al., 2006) pode ser visto na Figura 17. O *PIDPlus* mantém no último nível o sinal de controle calculado até o momento em que uma nova medida for recebida. Importante frisar que sua sintonia independe do período de amostragem, sendo dependente apenas das características físicas da planta.

Figura 17 – Estrutura do controlador *PIDPlus*.



Fonte: Song et al. (2006).

A realimentação (*newest measurement value*) e o filtro de 1° ordem (*filter*) são alterados para criar a contribuição de reposição tendo o seguinte comportamento:

- Manter a última saída do filtro calculado (I_{n-1}), até uma nova medição ser informada (*new value flag*);
- Quando uma nova medição é recebida (*new value flag*), utilizar a nova saída do filtro como contribuição da realimentação (I_n).

A principal diferença entre o PID e o *PIDPlus* está na parte integrativa que é substituído por um filtro de 1° ordem conforme (1).

$$I_n = I_{n-1} + (U_{n-1} - I_{n-1})(1 - e^{\frac{-\Delta T}{Treset}}) \tag{1}$$

$$\Delta T = t_n - t_{n-1} \tag{2}$$

Onde: I = termo integrativo atual, I_{n-1} = termo integrativo anterior, U_{n-1} = saída do controlador anterior, ΔT = intervalo de tempo desde que o último valor medido foi recebido, t_n = tempo atual, t_{n-1} = tempo anterior e T_{reset} = constante de tempo da planta somado ao tempo morto.

Levando em consideração a parte derivativa do controlador PID Tradicional e o *PIDPlus*, temos que no tradicional o divisor na parte derivativa seria o período (discretização do controlador), enquanto que, no algoritmo *PIDPlus* é o tempo entre duas medições recebidas com sucesso ΔT. Fica evidente que o algoritmo modificado produz uma ação derivativa menor do que o PID (SONG et al., 2006).

$$D = K_D(\frac{e_n - e_{n-1}}{\Delta T}) \tag{3}$$

Onde: D = termo derivativo, K_D = ganho derivativo, e_n = erro atual, e_{n-1} = erro anterior.

O microsserviço Controle PID4.0 tem como principal funcionalidade controlar processos de forma redundante, responsável por calcular através de um algoritmo de controle o sinal de controle que será aplicado no processo que desejamos controlar. Cada instância desse serviço tem como função implementar uma malha fechada de controle.

Dessa forma, por exemplo, o microsserviço Controle PID4.0-1 é responsável pelo controle da malha nº1, o Controle PID-2 pela malha 2 e assim sucessivamente conforme for necessário. Caso o serviço Controle PID4.0 possua múltiplas instâncias com o mesmo nome, isso significa que este controlador possui réplicas redundantes.

Sendo assim, por exemplo, se houverem (n) microsserviços Controle PID4.0 rodando em locais diferentes, isso significa que o controlador responsável pelo controle da malha 1 possui (n-1) réplicas em redundância. O microsserviço pode ser hospedado em diferentes plataformas como: sistema embarcado, Computador (PC) ou nuvem local industrial. Para realizar a operação é necessário operar em conjunto com outros microsserviços (DAQ) para obter as variáveis do processo que será controlado.

Importante salientar que o responsável por fazer a comunicação entre os microsserviços é o *transporter*, dessa forma, não importando o número de cópias (mesmo serviço com nome diferente) ou réplicas (várias instâncias do mesmo serviço) do controlador, toda comunicação é transparente e automática através do *transporter*.

4 DESENVOLVIMENTO E RESULTADOS

Em uma arquitetura a microsserviços, a aplicação é desenvolvida através da composição dos microsserviços desenvolvidos. Com essa composição é definido quais microsserviços serão utilizados e sua sequência de execução para obtermos a funcionalidade desejada. Essa composição pode ser feita via Orquestração (onde uma aplicação maestro comanda a execução dos serviços) ou via Coreografia (onde os serviços conhecem previamente a sequência de execução). Neste trabalho foi utilizado o formato de Orquestração.

O desenvolvimento dos códigos em *JavaScript* dos microsserviços foi realizado no Visual Studio Code. Uma extensão *SSHExtension*, foi instalada permitindo o acesso remoto ao console da *Raspberry Pi*, desta forma não é necessário utilizar periféricos como teclado, mouse e monitor conectados a *Raspberry*. Outra ferramenta importante utilizada foi o compartilhamento de pasta que permite acessar os arquivos pelo PC (Windows), facilitando o gerenciamento e permitindo que o desenvolvimento de todas as aplicações fossem concentradas em um único computador.

4.1 Microserviço PID4.0

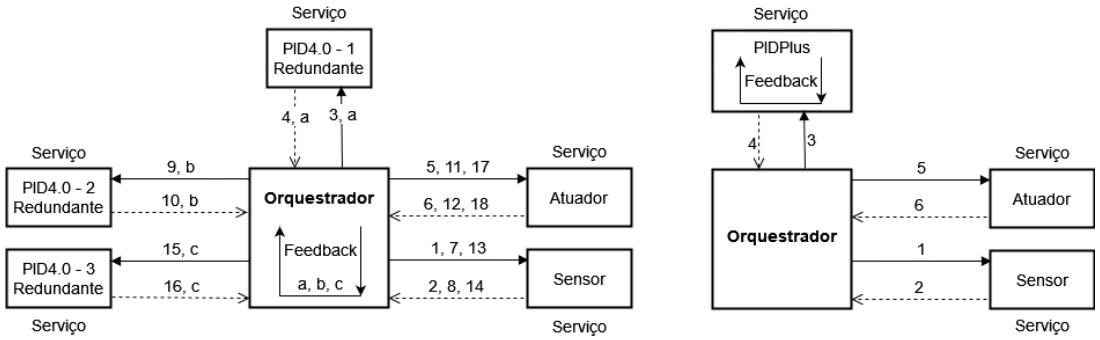
Para desenvolver o Controlador PID4.0 com redundância baseado no controlador *PIDPlus* demonstrado aqui temos que fazer algumas alterações em seu algoritmo e a principal delas é utilizar um registrador que possa armazenar algumas variáveis que serão necessárias quando o microsserviço PID4.0 for atuar e fazer os cálculos de controle.

Os termos das equações (1) e (3) são fundamentais para o desenvolvimento do serviço de controle PID4.0 com redundância. Para habilitar a operação do microsserviço, alguns desses itens das equações (1) e (3) devem ser armazenados em um registrador que é um *feedback* dos dados de controle conforme mostrado na Figura 18. Toda vez que o microsserviço for atuar é necessário que busque os dados no registrador e ao final do serviço que este seja atualizado. Os itens que devem ser armazenados são, erro = e, termo integrativo = I, saída do controlador = U e o tempo atual que será utilizado para calcular o ΔT do próximo serviço. O diferencial é que ao ser disponibilizado como um serviço, o microsserviço PID4.0 pode ser replicado em múltiplas plataformas como computador, nuvem ou sistema embarcado, gerando uma grande economia de *hardware*.

Para o desenvolvimento foi criado o diagrama de blocos apresentado na Figura 18, que mostra também a sequência de orquestração do PID4.0 e do PIDPlus , onde notamos que a principal diferença entre os dois é onde o *feedback* é feito com os dados de controle necessários para efetuar o cálculo. No caso do microsserviço PID4.0 o *feedback* dos dados de controle é feito dentro do orquestrador o que torna possível o compartilhamento desses dados entre todos os

serviços PID4.0 disponíveis, permitindo a redundância. Enquanto que para o caso do controlador PIDPlus o *feedback* dos dados de controle é feito dentro do próprio microsserviço o que não permite o seu compartilhamento, não permitindo a redundância.

Figura 18 – Diagrama sequencial de orquestração PID4.0 e PIDPlus.



Fonte: Autor.

Na Figura 18 apresentamos um exemplo de um diagrama sequencial, onde consideramos que o orquestrador irá chamar os microsserviços PID4.0 de forma sequencial, ou seja, primeiro o PID4.0- 1, depois o PID4.0- 2 e por fim o PID4.0-3. Para o primeiro ciclo de orquestração, o primeiro processo a ser realizado é a leitura feita pelo serviço de sensor, requisitado em 1 e respondido em 2 para o orquestrador. O próximo passo é enviar os dados dos sensores mais os últimos dados de controle representado por "a"($I_{n-1}, U_{n-1}, t_{n-1}$ e e_{n-1}) para o PID4.0- 1 através de (3, a). O PID4.0- 1 responde em (4, a) para o orquestrador que atualiza os dados de controle com o *feedback* e envia os dados em 5 para serviço atuador que por fim responde em 6 para o orquestrador.

O próximo ciclo seguirá os mesmos passos conforme diagrama sequencial de orquestração, começando em 7 e terminando em 12, porém desta vez os dados de controle ($I_{n-1}, U_{n-1}, t_{n-1}$ e e_{n-1}) que serão utilizados no cálculo pelo PID4.0- 2 são os que foram atualizados no ciclo anterior no *feedback* representado por "b". Por fim o último ciclo começa no passo 13 e finaliza no 18 seguindo o mesmo processo dos demais, porém agora os dados de controle atualizados pelo PID4.0- 2 no *feedback* representado por "c"que será enviado com o passo 15 para o PID4.0-3, que irá realizar o cálculo de controle e atualizar o *feedback* em 16.

Ainda na Figura 18 é apresentado o diagrama sequencial de orquestração do PIDPlus, é possível observar que a principal diferença do PID4.0 é que no PIDPlus o *feedback* dos dados de controle está alocado internamente no microsserviço de controle PIDPlus o que não permite o compartilhamento dos seus dados de controle, não permitindo a redundância. Seguindo a sequencia de orquestração de 1 a 6 é possível observar que após o cálculo de controle os dados serão atualizados localmente no *feedback* interno, o que torna necessário a utilização deste mesmo microsserviço para os demais ciclos. Desta forma se fizermos uma réplica do PIDPus

não será possível termos a redundância, pois os dados ($I_{n-1}, U_{n-1}, t_{n-1}$ e e_{n-1}) não estarão disponíveis para a réplica.

O PID4.0 implementado em *JavaScript* conta com uma ação que recebe da aplicação externa (LabVIEW) os parâmetros de controle de ganho proporcional (kp), tempo integrativo (ti), tempo derivativo (td), *setpoint*, variável de processo (PV), erro anterior (e_{n-1}), termo integrativo anterior (I_{n-1}), saída do controlador anterior (U_{n-1}) e o tempo anterior (t_{n-1}). Para armazenar os dados de algumas das variáveis das equações (1), (2) e (3) apresentado na seção 3.3, foi implementado um registrador (*feedback* dos dados de controle na Figura 18) dentro da aplicação externa LabVIEW .

As variáveis que devem ser armazenadas no registrador são erro (e_n), termo integrativo (I_n), saída do controlador (U_n) e o tempo atual(t_n) fornecido pelo sistema onde o microsserviço está embarcado e por este motivo é fundamental que o relógio desses locais onde estão os microsserviços PID4.0 estejam sincronizados. As variáveis armazenadas no registrador são atualizadas sempre que o serviço PID4.0 é requisitado, pois desta forma garantimos que este registrador tenha sempre guardado o ultimo valor dessas variáveis de controle ($I_{n-1}, U_{n-1}, t_{n-1}$ e e_{n-1}) que serão utilizadas na próxima vez que o serviço de controle atuar conforme equações (1), (2) e (3) apresentadas na seção 3.3.

Desta forma esse registrador se comporta como um *shift register* que é basicamente um dispositivo sequencial que carrega os dados atuais em suas entradas, e na sequência colocando em sua saída uma vez a cada ciclo. Os dados são armazenados em um formato tipo *array*, que tem como característica armazenar uma porção de elementos identificados por um índice. Importante frisar que o diferencial do serviço PID4.0 redundante em relação ao *PIDPlus* dedicado é justamente onde é feito o registro dessas variáveis de interesse, já que no *PIDPlus* é feito internamente no próprio serviço enquanto que no PID4.0 será feito na aplicação que faz a orquestração e dessa forma pode disponibilizar esses dados quando forem requisitados por qualquer um dos PID4.0 independente de onde estejam alocados, tornado a redundância possível.

Este trabalho se baseia na arquitetura MOA, por isso foi desenvolvido um controlador redundante de processo no formato de um microsserviço. Para isso foi necessário uma adaptação ou nova concepção de um algoritmo de controle tradicional, pois este tipo de arquitetura trabalha conectada em rede, necessitando propor um algoritmo de controle via rede.

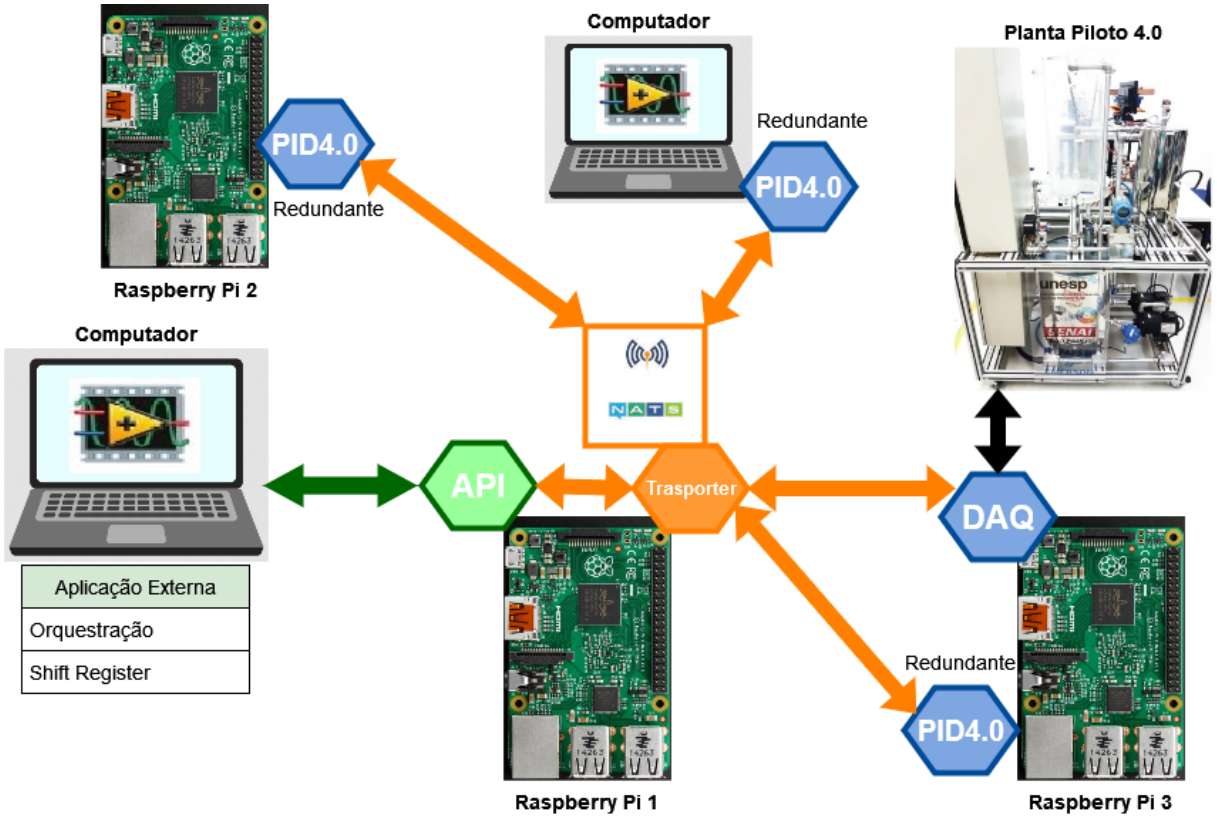
O microsserviço de controle PID4.0 faz o controle do processo com o acesso feito por requisições na rede, desta forma o algoritmo foi modificado para operar através de requisições do *framework* Moleculer e desenvolvido em linguagem *JavaScript*. Sendo assim o controlador proposto pode ser utilizado como um serviço do Moleculer, podendo ser requisitado por qualquer outro serviço ou aplicação interna ou externa.

4.2 Testes e Validação da Redundância

Para a etapa de testes foram utilizadas três *Raspberry Pi*, o computador e a planta piloto 4.0. Na *Raspberry Pi* 3 foi alocado o microsserviço PID4.0 e o DAQ, lembrando que o DAQ faz a comunicação com os sensores e atuadores da planta. A *Raspberry* 2 foi alocado um microsserviço PID4.0 redundante, na *Raspberry* 1 foi alocado os microsserviços de comunicação da planta, o API que faz a comunicação externa e o Transporter que faz a interna. No computador também foi alocado um PID4.0 redundante e o aplicativo *LabView* que faz a orquestração e registro das variáveis de controle no *shift register* conforme Figura 19 .

A Figura 19 faz uma ilustração de como os serviços estão alocados e distribuídos na planta para realização dos testes, desta forma conseguiremos demonstrar a função de cada serviço e como ele atua em todo o processo.

Figura 19 – Disposição dos serviços.



Fonte: Autor.

Na Figura 19 a parte representada pela aplicação externa é responsável por fazer a orquestração dos serviços, bem como o registro das variáveis utilizadas no controle e realizar a monitoração do processo. Essa aplicação foi desenvolvida utilizando o *software* LabVIEW. Como essa aplicação é externa sua comunicação com os demais microsserviços é feita utilizando o padrão REST via o microsserviço API Gateway representado pelo hexágono verde e embarcado na Raspberry Pi 1. A aplicação externa pode ser feita e desenvolvida em diferentes plataformas

de *software* já consolidadas e tradicionais na indústria, o que deve gerar maior flexibilidade em seu desenvolvimento.

O hexágono laranja representa o serviço de *transporter* e este faz a comunicação entre os microsserviços e entre estes e as possíveis aplicações externas. Está embarcado na Raspberry Pi 1 e faz a comunicação entre os serviços utilizando o NATS. O microsserviço DAQ representado por um hexágono azul está embarcado na Rasberry Pi 3 e é responsável por fazer a aquisição de dados do processo conforme já mencionado.

O serviço de controle PID4.0, foco de estudo deste trabalho, está representado pelo hexágono azul e tem um serviço alocado na Rasberry Pi 3, uma replica no computador e outra na Raspberry Pi 2. Este serviço tem a função de fazer o cálculo de controle e atualizar as variáveis de controle no *shift register*, desta forma permitindo a redundância dos controladores embarcados em *hardwares* diferentes.

Neste trabalho contamos com duas aplicações, a de monitoramento e a de controle com redundância. A aplicação de monitoramento faz a aquisição e monitoramento de dados das malhas do processo com o MOA. Os dados de processos são adquiridos através do serviço DAQ. Essa aplicação tem o objetivo de fazer uma interface para o monitoramento das variáveis de processo, contando com uma análise de comportamento e desempenho ao longo do tempo da ferramenta de controle. A aplicação de controle tem como responsabilidade a efetivação do controle em malha fechada das variáveis de interesse, através da composição dos serviços DAQ e PID4.0, além de sua função fundamental de redundância.

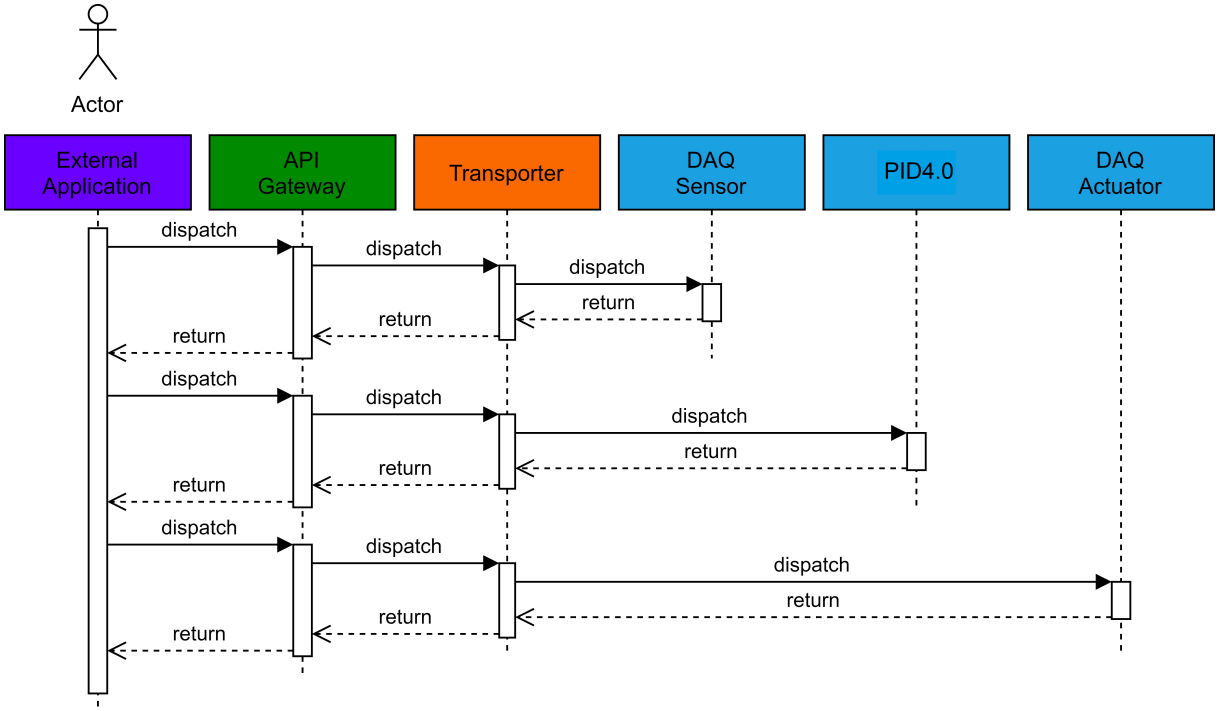
Na Figura 20 apresentamos as etapas dos microsserviços, feita por uma aplicação externa no nosso caso feita através do *software* LabVIEW. Neste diagrama é possível verificar a sequência de execução dos microsserviços definida em cada aplicação. Neste passo a passo é possível observar todo o caminho realizado pelos pacotes de comunicação desde a aplicação externa até o controle e sua atuação no atuador que será apresentado por etapas:

1. Ao termos uma solicitação externa a mesma será enviada para o *API Gateway*, que encaminhará ao *Transporter* e este encaminhará para o serviço DAQ que fará a leitura dos dados do processo, retornando a aplicação externa.

2. A aplicação externa enviará esses dados para o microsserviço de controle *PID4.0* via *API Gateway* que encaminhá ao transporter e este encaminhará para o serviço de controle que fará os cálculos e retornará a aplicação externa.

3. Por fim a aplicação externa novamente irá passar os dados via *API Gateway* que transmitirá para o transporter que enviará para o serviço DAQ que atualizará os atuadores do processo e depois retornará a aplicação externa, finalizando o a sequência de orquestração do ciclo de controle.

Figura 20 – Sequência de comunicação dos Microsserviços.



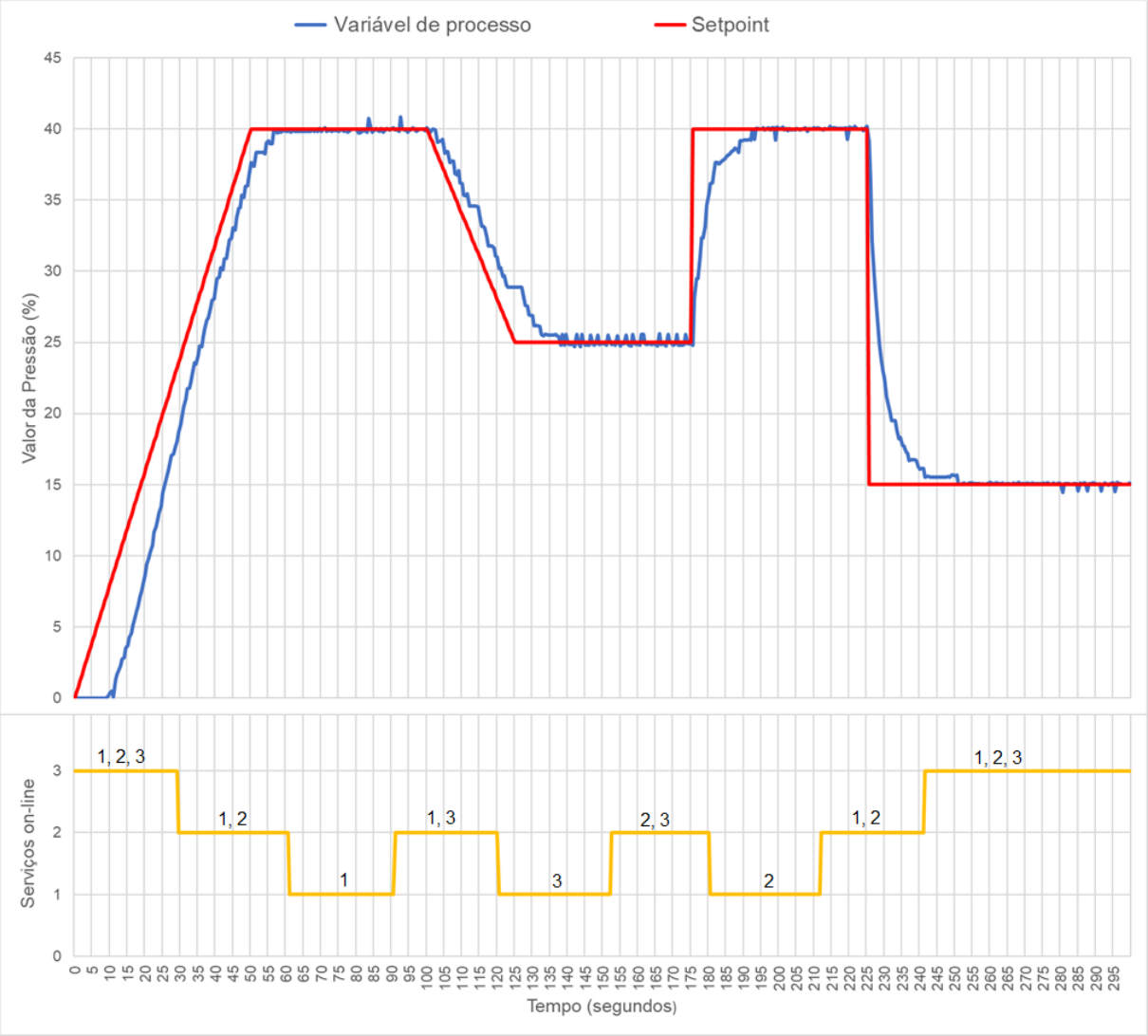
Fonte: Autor.

Para validarmos a redundância do controlador PID4.0, foi proposto neste trabalho dois experimentos que foram realizados na planta piloto 4.0. O primeiro experimento será feito com os três serviços on-line e com o passar do tempo iremos deixar alguns serviços off-line, demonstrando que o controle é feito independentemente da configuração e da quantidade de serviços on-line. O segundo experimento será uma comparação entre o controle PIDPlus que funciona de forma dedicada sem a redundância e o controlador PID4.0 que funciona da mesma forma, porém permitindo a redundância.

Com o intuito de validarmos o microsserviço de controle PID4.0 com redundância fizemos um experimento de controle de pressão de linha na planta piloto 4.0 com os serviços 1, 2 e 3. Neste experimento contamos com três serviços que são capazes de realizar o controle do processo individualmente ou com qualquer configuração, todos os serviços on-line, dois ou apenas 1.

Para demonstrar a operação da redundância de controle da malha de processo, o experimento começou com todos os serviços on-line e a cada 30 segundos geramos uma mudança na quantidade de serviços on-line de forma aleatória e essas etapas do experimento podem ser vistas no gráfico da Figura 21, onde a linha em amarelo representa em cada intervalo de tempo quantos serviços estão on-line em cada instante, e acima dessa linha o número de quais serviços que estão on-line neste período, deste modo conseguimos simular uma possível falha em todos os controladores.

Figura 21 – Controle malha de pressão.



Fonte: Autor.

O gráfico da Figura 21, traça o perfil de Setpoint em vermelho e o da Variável de Processo em azul referente a malha de pressão da planta piloto. Com isso verificamos que o processo foi controlado conforme esperado e demandado pelo perfil de Setpoint, consolidando a eficácia do controle redundante realizado pelo controlador PID4.0. Este controle foi realizado no processo através do algorítimo desenvolvido em *JavaScript*, e contando com um registrador para armazenar os dados de controle que poderão ser utilizados por qualquer um dos serviços de controle conforme demandado.

Na Figura 21 , através da linha em amarelo que mostra a quantidade de serviços on-line, temos uma análise que demonstra melhor como foi feito o experimento para consolidar a redundância do controlador PID4.0, ressaltando que os serviços foram colocados em modo on-line ou off-line manualmente. Neste gráfico podemos observar por exemplo que nos primeiros 30 segundos

temos os três serviços online representado pela linha amarela e acima desta linha quais serviços especificamente estão on-line naquele momento. Após os 30 segundos colocamos o serviço 3 em off-line, representado pela linha amarela que naquele momento de 30 a 60 segundos tínhamos 2 serviços on-line o 1 e 2. Passado mais 30 segundos colocamos o serviço 2 em off-line, restando somente o serviço 1 para realizar o controle e desta forma fomos alterando a quantidade de serviços ao longo do experimento conforme pode ser visto na Figura 21.

Com este experimento conseguimos simular uma possível falha em qualquer um dos serviços de controle, forçando que o mesmo fique off-line e é possível observar que a função de controle continua funcionando perfeitamente durante todo o processo, não tendo nenhuma influência a quantidade de serviços on-line naquele momento, ou seja podemos ter diversos serviços redundantes on-line ao mesmo tempo.

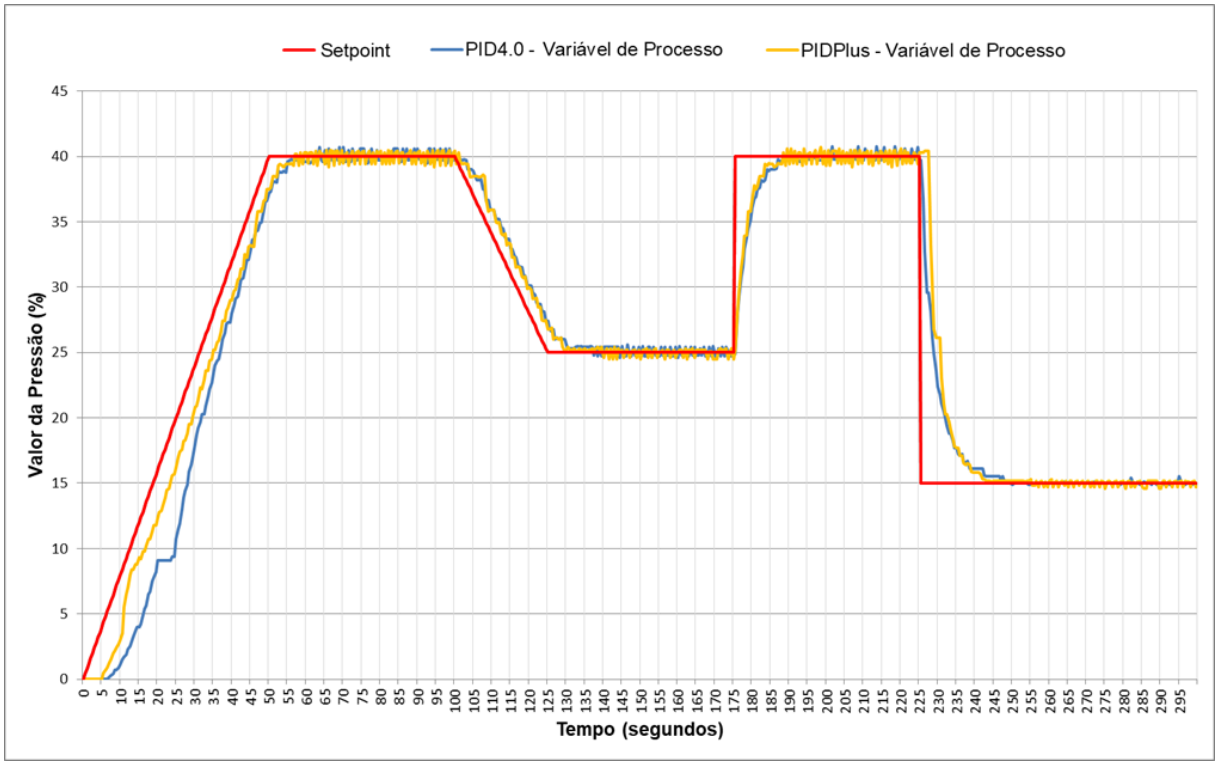
Este experimento consolidou a eficácia do controlador PID4.0 em fazer o controle do processo de forma redundante. Importante observar que neste experimento tivemos oito alterações na quantidade de serviços on-line durante todo o experimento, o que poderia ter gerado uma perturbação no controle do processo, porém conforme podemos observar em Figura 21 representado pela linha em azul, não tivemos nenhum transiente no perfil de controle do processo em nenhum momento que tivemos uma alteração da quantidade de serviços on-line representado pela linha em amarelo, o que demonstra que a falha em qualquer um dos controladores é transparente para o controle do processo.

O segundo experimento realizado, foi a comparação entre o controlador PIDPlus utilizado para fazer o controle de todas as malhas da planta, mas de forma dedicada e sem redundância e o controlador PID4.0 que faz o mesmo tipo de controle, seguindo os mesmo princípios porém de forma redundante. Utilizamos a malha de de pressão e o resultado será apresentado na Figura 22.

O gráfico da Figura 22 apresenta a linha em vermelho o perfil de Setpoint proposto tanto para o controlador PIDPlus, como para o PID4.0. A linha em azul representa o perfil traçado pelo controle da variável de processo da malha de pressão utilizando como controlador o PID4.0. A linha em amarelo representa o perfil traçado pelo controle da variável de processo da malha de pressão utilizando como controlador o PIDPlus.

Como pode ser visto na Figura 22 os dois controladores foram eficientes e o processo foi controlado conforme esperado e demandado pelo perfil de Setpoint em vermelho. Não há diferença significativa entre o perfil traçado na linha em azul pelo controlador PID4.0 e o perfil traçado na linha em amarelo pelo controlador PIDPlus, desta forma podendo concluir que os dois conseguem responder de forma satisfatória a demanda do processo, porém com uma vantagem para o serviço PID4.0 que é feito de forma redundante, podendo ter seu serviço alocado em diversos ambientes, o que trará maior segurança e confiabilidade ao processo, permitindo uma operação contínua, facilitando o planejamento das manutenções e uma significativa economia da operação.

Figura 22 – Comparação PID4.0 e PIDPlus.



Fonte: Autor.

Após a realização desses dois experimentos podemos concluir que a aplicação de controle com redundância PID4.0 funciona conforme esperado, atendendo todos os requisitos necessários para realizar o controle dos processos da planta piloto de forma redundante.

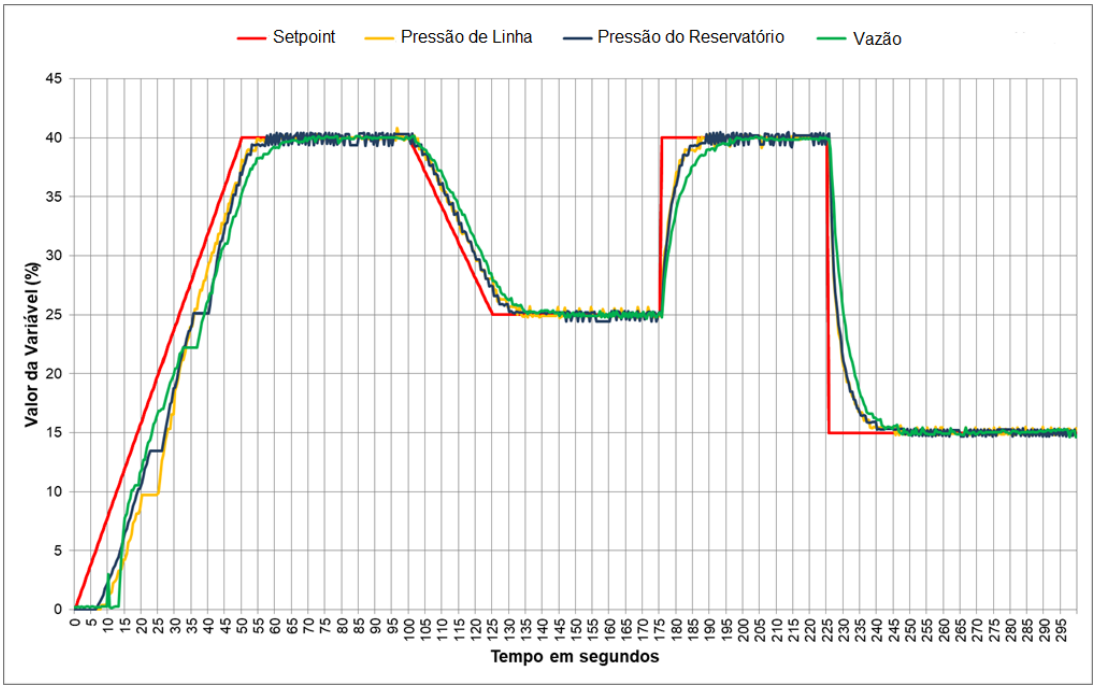
Depois de validado o controlador PID4.0, aprofundamos na análise de seu funcionamento, sendo assim foi realizado experimentos de validação deste controle em outras malhas de controle disponíveis na planta piloto 4.0. As malhas são vazão da linha, pressão da linha e pressão do reservatório R01 apresentado na Figura 23, temos disponível também a malha de nível, porém esta não foi utilizada porque apresenta uma dinâmica mais lenta e não daria para comparar com este mesmo perfil. Além disso também foi analisado o experimento com duas malhas de controle sendo executadas ao mesmo tempo conforme Figura 24.

Conforme aFigura 23, onde temos em vermelho o perfil de setpoint, em amarelo a malha de pressão de linha, em azul a malha de pressão do reservatório e em verde a malha de vazão, todas as malhas de controle testadas realizaram o controle conforme esperado e demandado pelo perfil de setpoint validando este controlador para essas três malhas.

Outro importante experimento realizado para validação deste controlador foi executar duas malhas ao mesmo tempo, para este caso colocamos duas aplicações externas rodando ao mesmo tempo, ou seja tínhamos dois orquestradores atuando. Escolhemos a malha de pressão da linha em verde que tem como atuador a B02 e a malha de pressão do reservatório em azul que tem

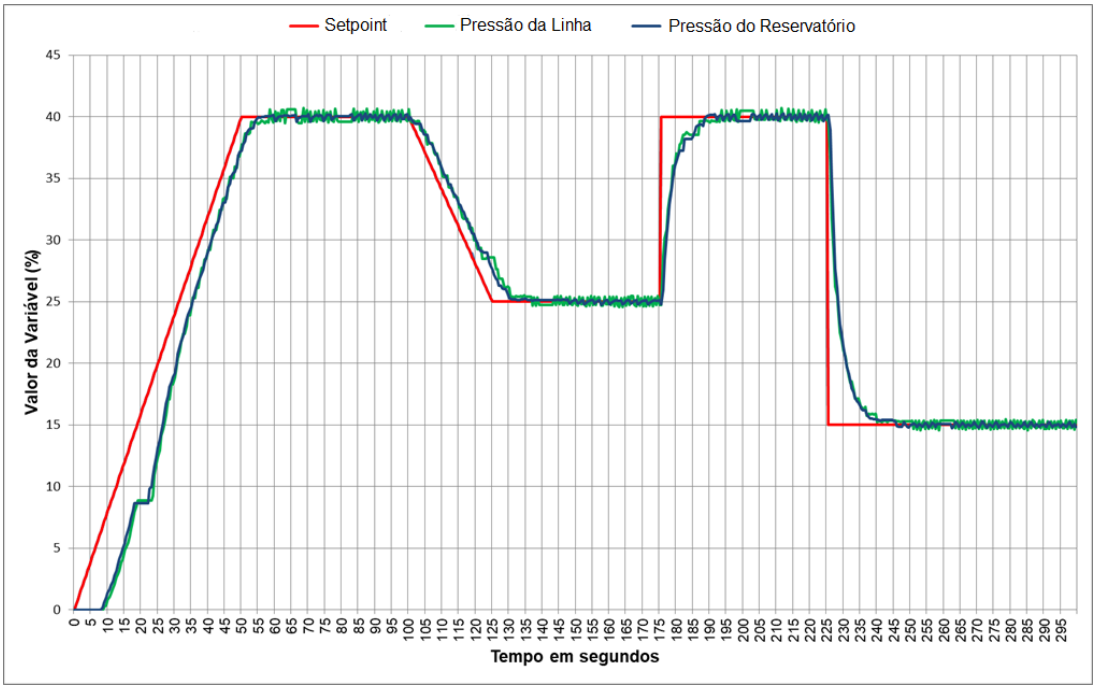
como atuador a B01, justamente para não termos a interferência do controle de uma malha em outra indevidamente.

Figura 23 – Malhas de controle.



Fonte: Autor.

Figura 24 – Controle de duas malhas ao mesmo tempo.



Fonte: Autor.

Como podemos verificar na Figura 24, apesar de termos uma demanda maior de dados, as duas malhas foram controladas conforme esperado e demandado pelo perfil de setpoint validando este experimento.

4.2.1 Comparação de desempenho dos controladores PIDPlus e PID4.0

Na metodologia de projeto de um controlador PID, um dos critérios de desempenho mais utilizados é o erro, que é a diferença entre o sinal da saída da planta e o sinal de setpoint. Usando este critério de erro como parâmetro para comparação de desempenho entre os controladores, usaremos para análise as funções baseadas nestes erros que são Integrated squared Error (ISE), Integrated Absolut Error (IAE), Integrated Timed Absolut Error (ITAE) e Integrated Timed Squared Error (ITSE) conforme apresentado abaixo.

$$ISE = \int_0^\infty e^2(t)\,dt \tag{1}$$

$$IAE = \int_0^\infty |e(t)|\,dt \tag{2}$$

$$ITAE = \int_0^\infty t.|e(t)|\,dt \tag{3}$$

$$ITSE = \int_0^\infty t.e^2(t)\,dt \tag{4}$$

A integral do erro quadrático (ISE), eleva o erro ao quadrado de forma a maximizar os erros de maior amplitude e minimizar os de menor amplitude. A integral do erro absoluto (IAE), leva em consideração somente o módulo do erro ao longo do tempo. A integral do tempo multiplicado pelo erro absoluto (ITAE), onde além de levar em consideração o erro absoluto também leva em consideração o tempo, ou seja, os erros após um longo período de inicialização do sistema de controle são mais penalizados. Da mesma forma a integral do tempo multiplicado pelo erro quadrático (ITSE), além de levar em consideração o erro quadrático também sofre uma penalização maior conforme maior o tempo de controle.

Outro indicador que iremos utilizar para aumentar a robustez de nossa análise de comparação é o que chamaremos de Índice de desempenho (Id), que avaliará o desempenho em porcentagem do resultado ideal, e para este item o quanto mais próximo de zero melhor foi o desempenho do controlador e será calculado conforme fórmula apresentada abaixo.

$$Id = \frac{IAE}{\int_0^T SP(t)\,dt} \tag{5}$$

Onde temos como parâmetros: T = período de execução do experimento, SP = setpoint requerido e IAE = Integral do valor absoluto do erro conforme equação 2.

Após realizarmos os cálculos de desempenho dos controladores apresentamos o resultado na tabela 1, podemos concluir que temos um desempenho semelhante no controle do processo realizado pelo controlador PIDPlus de forma dedicada ou pelo controlador PID4.0 de forma redundante. Se observarmos detalhadamente concluiremos que o PID4.0 leva uma pequena vantagem após um longo período de inicialização do sistema de controle, pois obteve números melhores, ou seja mais baixos no ITAE e ITSE.

Tabela 1 – Desempenho dos controladores

Critério	PIDPlus	PID4.0
IAE	798,43	868,03
ITAE	90302,75	89534,56
ISE	4870,89	5022,99
ITSE	739622,15	665647,57
Id	4,88%	5,3 %

Fonte: Autor.

Com isso podemos concluir que é vantajoso utilizarmos o controlador PID4.0, pois além de termos a vantagem de realizar o controle de forma redundante, podendo ser replicado em diferentes componentes, o mesmo tem um desempenho equivalente ao controlador PIDPlus no controle do processo.

4.2.2 Comparação do tempo de Controle do PIDPlus e PID4.0

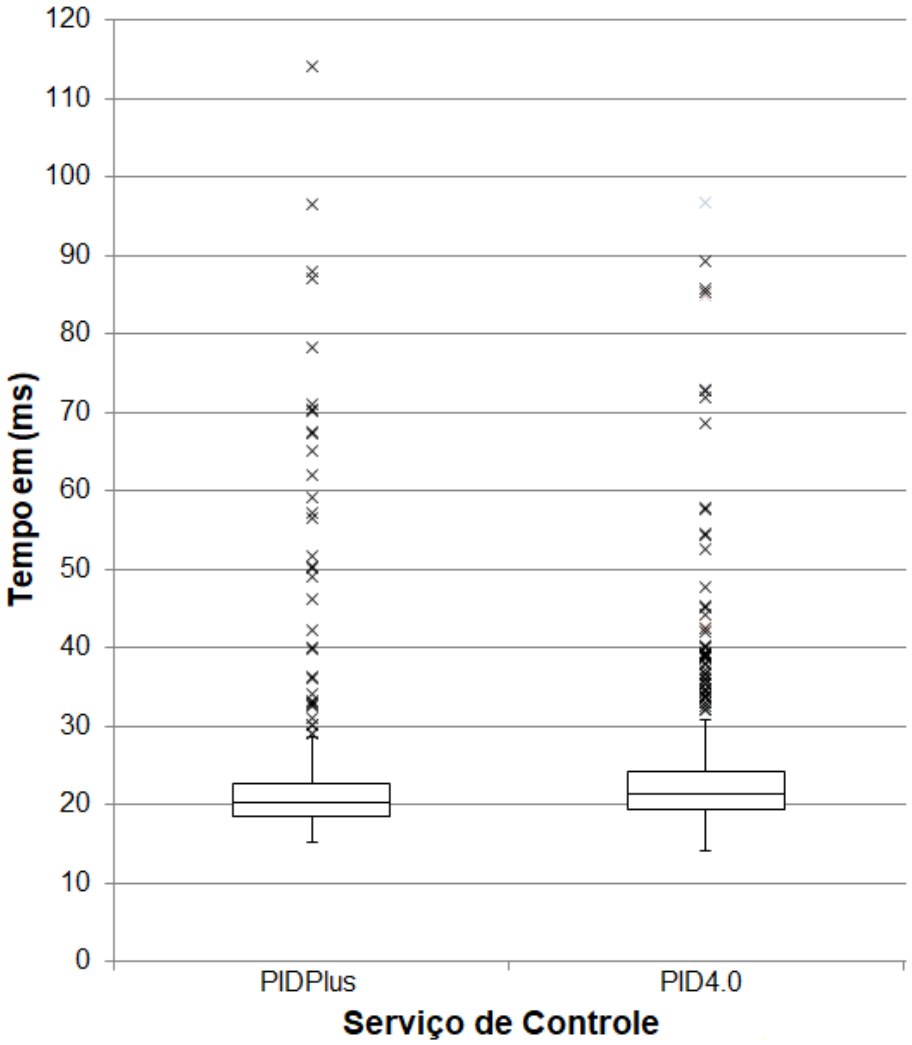
Visando analisar a diferença de tempo em milissegundos entre os controladores PIDPlus e PID4.0, iremos utilizar a ferramenta de diagrama de caixa conhecido como boxplot, que nos permite visualizar a distribuição e valores discrepantes (outliers) dos dados. Para isso iremos fazer duas análises, primeiro levando em consideração somente o tempo do controlador e depois levando em consideração o tempo total de orquestração do processo, ou seja o ciclo de controle, levando em consideração o tempo do serviço DAQ de leitura dos sensores e atuação nos atuadores conforme diagrama na Figura 25. Importante frisar que essas amostras de tempo foram retiradas dos próprios experimentos realizados neste trabalho.

Analisando o comportamento no diagrama da Figura 25, podemos concluir que os dois controladores tem comportamento de tempo parecidos, sendo que o controlador PIDPlus leva uma pequena vantagem, conforme esperado já que tem menos dados a serem processados. O PIDPlus tem uma mediana de 20,3 ms e concentra 50% dos valores entre 18,6 a 22,6 ms, 25% entre 15,2 e 18,6 e os outros 25% entre 22,6 e 28,5 ms, o que demonstra que tem boa simetria e baixa dispersão. O PID4.0 tem uma mediana de de 21,4 ms e concentra 50% dos valores entre 19,4 a 24,1 ms, 25% entre 13,9 e 19,4 e os outros 25% entre 24,1 e 30,7 ms, o que demonstra que

também tem boa simetria e baixa dispersão, porém com valores em ms um pouco acima do que os encontrados no PIDPlus. Outro ponto importante a se observar é que o controlador PID4.0 obteve um número superior de outliers.

Comparando esses dois modelos de controladores, podemos concluir que os dois tem um comportamento parecido em relação ao tempo gasto pelo controlador realizar o seu processo, com uma diferença de apenas 1,1 ms entre as medianas e concentrando a maior parte dos valores em uma diferença parecida entre primeiro e terceiro quartil, porém com uma pequena vantagem para o PIDPlus.

Figura 25 – Comparação de tempo dos Controladores



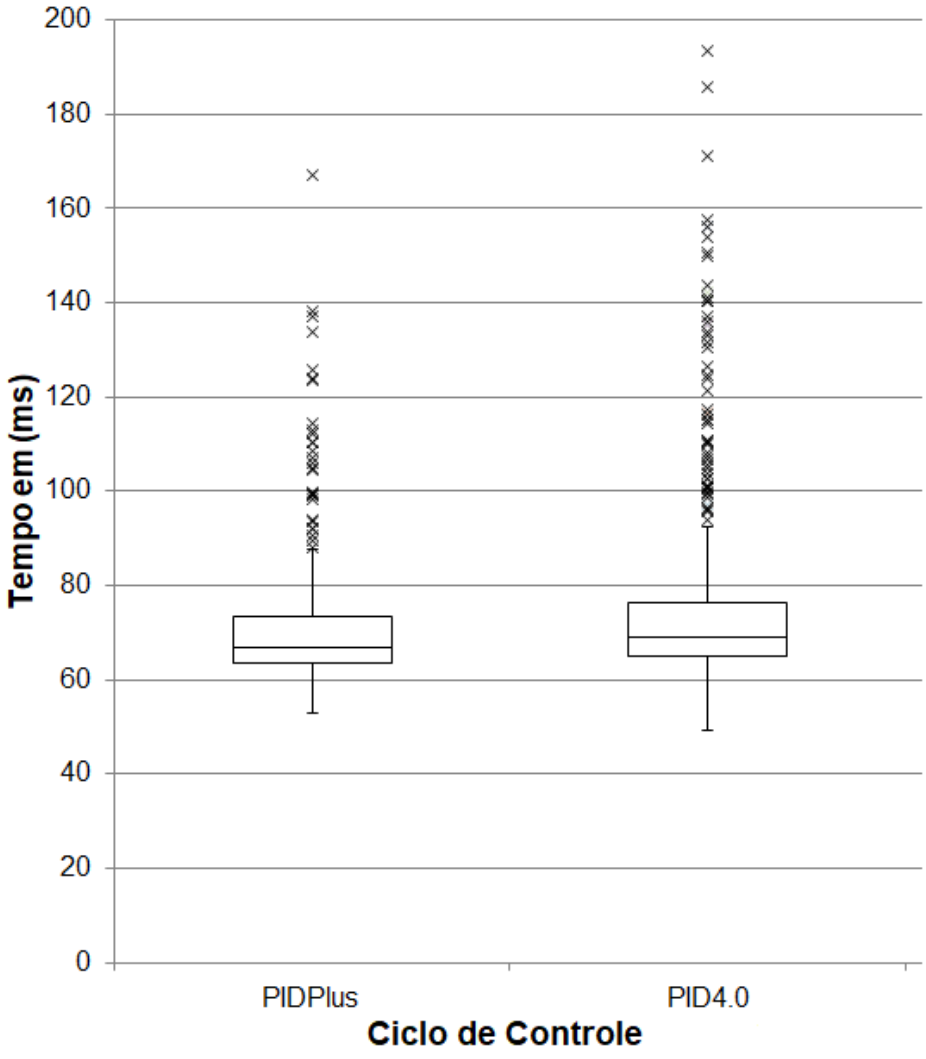
Fonte: Autor.

Levando em consideração agora o ciclo de controle, desde a leitura dos sensores até a atuação nos atuadores feitos pelo serviço DAQ, temos o diagrama da Figura 26. Podemos concluir que o comportamento de todo o processo é parecido com a análise anterior, concluindo que o PIDPlus

tem uma pequena vantagem em relação ao PID4.0.

Analisando o comportamento no diagrama da Figura 26, podemos concluir que os dois processos tem um comportamento no tempo parecidos, sendo que o controlador PIDPlus leva uma pequena vantagem, porém não tivemos uma alteração significativa no processo como um todo comparando os dois controladores. Comparando o ciclo de controle com o tempo só do controlador é possível concluir que a diferença de tempo encontrada entre os controladores é praticamente replicada para o processo como um todo, tendo o PID4.0 um tempo maior e com mais outliers, porém com boa simetria e baixa dispersão.

Figura 26 – Comparação de tempo do ciclo de controle



Fonte: Autor.

O ciclo de controle que tem como controlador o PIDPLus tem uma mediana de 66,9 ms e concentra 50% dos valores entre 63,5 a 73,3 ms, 25% entre 52,9 e 63,5 e os outros 25% entre 73,3 e 87,8 ms, o que demonstra que tem boa simetria e baixa dispersão. O ciclo de controle que

tem como controlador o PID4.0 tem uma mediana de 69,1 ms e concentra 50% dos valores entre 65,2 a 76,2 ms, 25% entre 49,1 e 65,2 e os outros 25% entre 76,2 e 92,2 ms, o que demonstra que também tem boa simetria e baixa dispersão, porém com valores em ms levemente superiores do que os encontrados no processo controlado pelo PIDPlus. Outro ponto importante a se observar é que o processo controlado pelo PID4.0 obteve um número superior de outliers.

Após a comparação de desempenho e tempo dos controladores, podemos concluir que os dois se comportam de forma semelhante, possuindo as mesmas características técnicas de controle e que os processos controlados por eles tem como resultado a mesma resposta de controle. Sendo assim o controlador PID4.0 tem uma vantagem em ser utilizado pois pode ser replicado em diversos componentes trazendo robustez, segurança, disponibilidade e tolerância a falhas.

4.3 Estratégias de Balanceamento de Requisição de Serviços Redundantes

Uma importante configuração fornecida pelo *Moleculer* a ser considerado é o balanceamento da rede, que define a política de requisição dos serviços existentes pelo Transporter. No caso dos controladores redundantes desse trabalho, essa política define qual dos serviços (original ou réplicas) serão requisitados para operação em cada iteração da malha de controle. Os tipos possíveis de configuração são Round-Robin strategy, Random strategy, CPU usage-based strategy e Latency-based strategy:

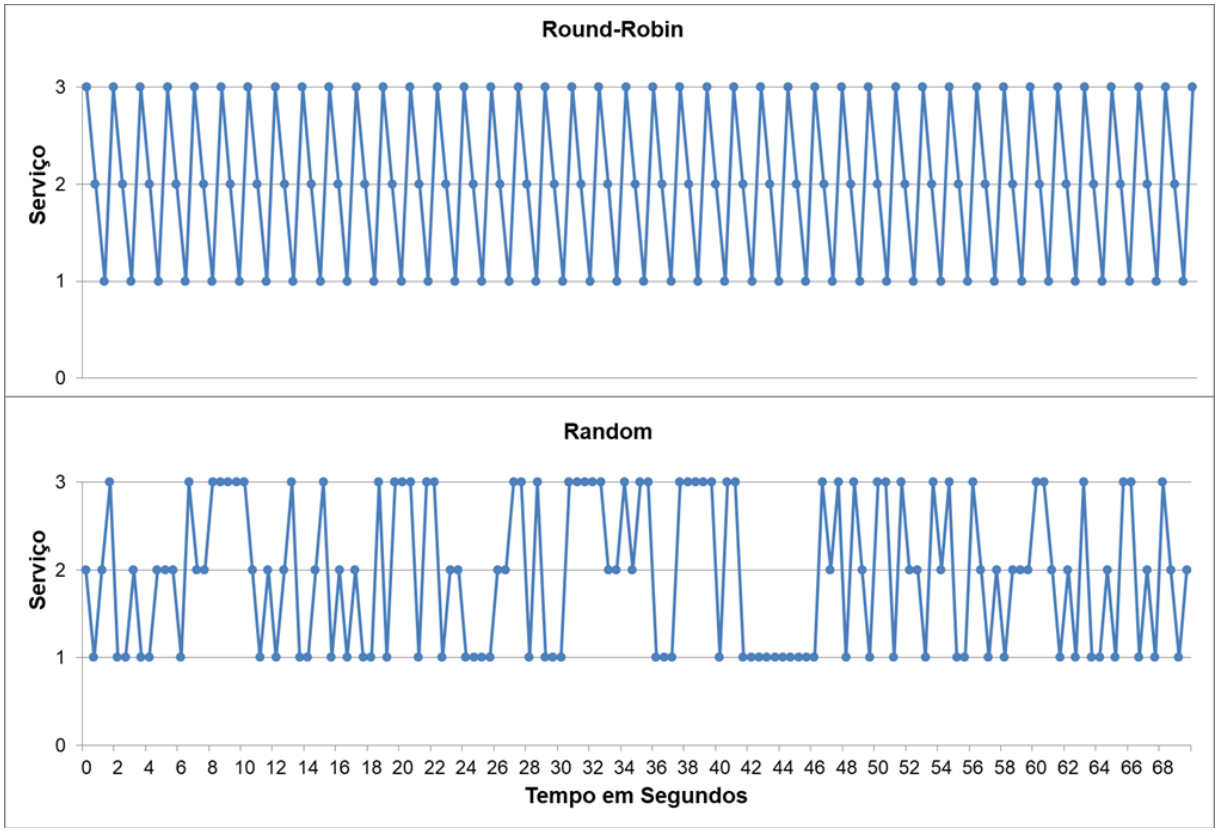
- Round-Robin strategy - Com essa estratégia o nó é selecionado através de um algoritmo round-robin que vai chamando os serviços de forma sequencial, conforme sequência que foram colocados on-line por exemplo $(1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 3)$.
- Random strategy - Utilizando essa estratégia o nó é selecionado de forma aleatória.
- CPU usage-based strategy - Utilizando essa estratégia o nó selecionado será o que tiver a menor utilização da CPU. Se tivermos uma grande quantidade de nós, essa estratégia irá fazer uma amostra e direcionar para o nó com menor uso de CPU, ao invés de fazer uma analise na CPU de todos os nós.
- Latency-based strategy - Com essa estratégia o nó que tiver a menor latência será o selecionado, que será medido através de comandos de ping feitos periodicamente. Também se tivermos muitos nós será feito uma amostra e será selecionado o de menor latência.

Importante salientar que para o experimento representado pela Figura 21 foi utilizado a estrategia Round-Robin o que significa que durante cada intervalo de tempo os serviços de controle foram requisitados de forma sequencial conforme os disponíveis naquele momento, por exemplo nos primeiros 30 segundos foi feito da seguinte forma $3 \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 1$.

Para verificarmos todas as estratégias de balanceamento da rede fizemos experimentos para cada uma das estratégias, o que será demonstrado através de gráficos. Para as estratégias

Round-Robin e Random o experimento foi realizado com os três serviços on-line e conforme podemos observar na Figura 27 para os primeiros 70 segundos, o primeiro gráfico da figura representa o Round-Robin, pois como podemos verificar no gráfico é seguido uma sequência ($3 \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 1$) ao longo de todo o período confirmando que funciona corretamente quando configurado.

Figura 27 – Estrategia Round-Robin e Random.



Fonte: Autor.

O segundo gráfico da Figura 27 representa a configuração no modo Random, que tem como estratégia escolher de forma aleatória entre todos os serviços disponíveis naquele período o que podemos confirmar através do gráfico seu formato aleatório.

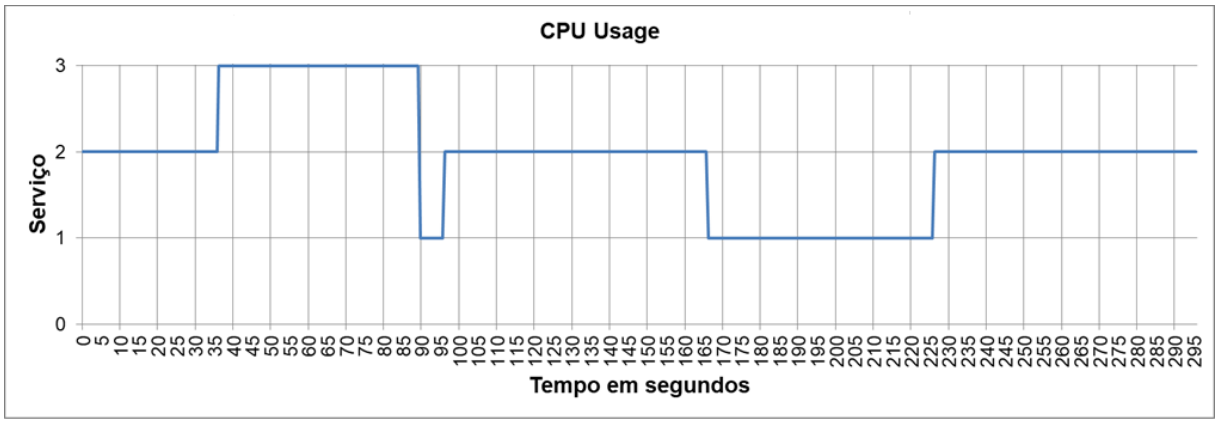
Para o experimento da estratégia CPU Usage, que verifica a porcentagem de uso de cada CPU dos serviços disponíveis e escolhe o de menor uso, foi instalado um programa que gera um stress na CPU de cada um dos *hardwares*. Começamos o experimento com os três serviços on-line e durante este período foi requisitado o serviço 2 porque a CPU do seu *hardware* estava com a menor porcentagem de uso, porém após 35 segundos onde o programa que gera o stress na CPU do serviço 2 e 1 foi efetivado conforme Figura 28.

É possível observar que por todo o período em que o programa estava em funcionamento o serviço escolhido foi o 3 que estava com menor uso da CPU. De 90 a 95 segundos tivemos um momento de transição para ajustar o programa de stress da CPU nos serviços 1 e 3. A partir

dos 95 segundos, após consolidado o stress nas CPUs dos serviços 1 e 3 temos a utilização do serviço 2 que estava com menor uso da CPU de 95 até 165 aproximadamente conforme gráfico Figura 28.

Em 165 segundos foi retirado o stress do serviço 1 e colocado no serviço 2 e 3 fazendo com que o serviço escolhido durante este período fosse o 1 que tinha o menor uso da CPU. Por fim em 255 segundos foi retirado o stress de todas as CPUs e o serviço escolhido foi o 2 até o final do experimento pois era o que tinha o menor uso da CPU conforme demonstrado no gráfico Figura 28.

Figura 28 – Estrategia CPU Usage



Fonte: Autor.

Desta forma é possível verificar na Figura 28 que a estratégia CPU Usage funciona conforme esperado e toda vez que geramos um stress na CPU de onde os serviços estão alocados o serviço utilizado é aquele que está sem o stress e consequentemente com o menor uso da CPU.

Por fim para a estratégia de latência que tem grande importância para este tipo de sistema baseado em serviços foi observado que com os três serviços on-line a tendência era escolher o serviço 3 que estava embarcado no computador e ligado a rede via cabo que tem menor latência do que os serviços 1 e 2 que estão embarcados nas *Raspbarry PI* e estão ligados à rede via wi-fi.

Para esta estratégia não foi possível verificar com certeza que o serviço escolhido era o de menor latência, porque não foi possível segregar a rede e sobrecarregar somente o ponto onde queríamos realizar o teste, pois ao sobrecarregar em qualquer ponto afetava a rede como um todo e não trouxe o resultado esperado. Importante ressaltar que para todas as estratégias de balanceamento de rede testadas a curva da variável de processo foi semelhante a apresentada na Figura 21.

Em aplicações de automação e controle, a comunicação em rede entre os serviços realizada pelo Transporter pode impactar no desempenho das malhas de controle. O Moleculer possui alguns tipos de *transporters* disponíveis a serem utilizados como TCP, NATs, Redis, MQTT, AMQP 1.0 e Kafka Transporter. Para este trabalho utilizamos o NATs transporter, mas para trabalhos futuros uma análise detalhada da influência de cada um desses tipos será importante.

4.3.1 Comparação de desempenho e tempo entre as estratégias de balanceamento de Requisição de serviços

Visando expandir nossos testes de validação iremos apresentar as diferenças encontradas no controlador PID4.0 quando colocado para operar em diferentes configurações de estratégias de balanceamento e requisição de serviços, utilizando a malha de pressão de linha.

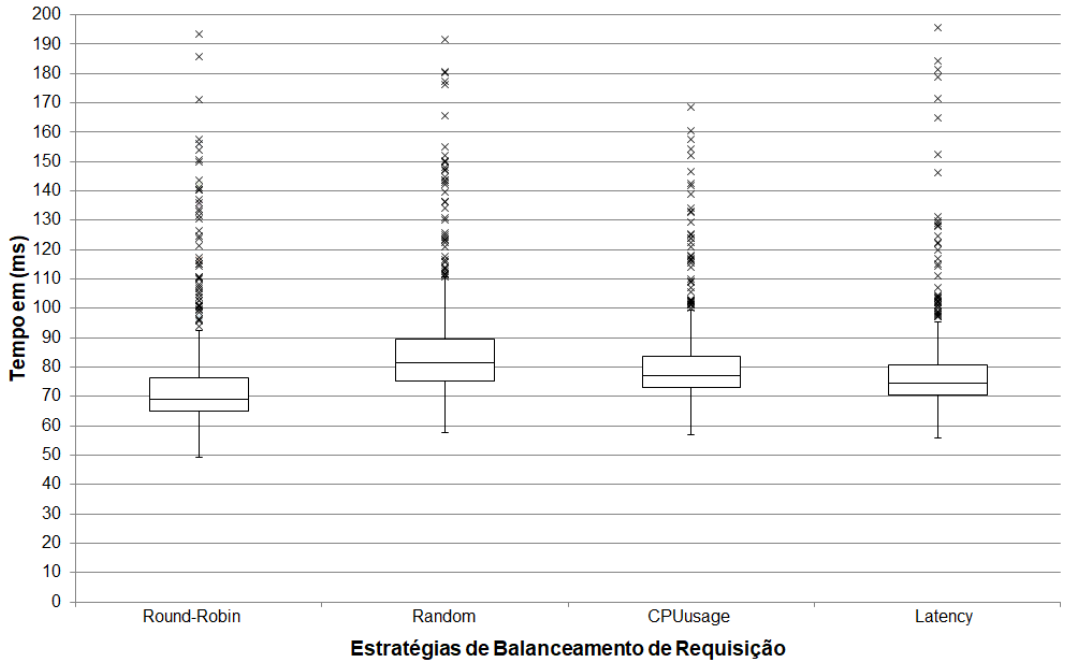
Iremos apresentar os dados referente ao desempenho dessas estratégias, sendo elas a Round-Robin, Random, Uso da CPU e Latência. Vale ressaltar que quando configuramos uma estratégia a mesma é utilizada para a comunicação entre todos os serviços utilizados pela planta, sendo assim essa configuração é feita no serviço de API Gateway e feito isso todos os demais serviços passam a utilizar essa configuração de comunicação para requisição dos serviços. Os dados de desempenho dessas estratégias são apresentados na tabela 2.

Tabela 2 – Desempenho das Estratégias de Requisição

Critério	Round-Robin	Random	Uso da CPU	Latência
IAE	868,06	890,74	863,60	883,83
ITAE	89534,56	92775,57	90662,48	90038,30
ISE	5022,99	5377,26	5007,80	5185,75
ITSE	665647,57	750707,13	679121,66	682453,82
Id	5,3%	5,44%	5,27%	5,4%

Fonte: Autor.

Figura 29 – Comparação de tempo entre as estratégias de requisição



Fonte: Autor.

Conforme pode ser observado na tabela 2, as duas estratégias que tiveram o melhor desempenho foram a Round-Robin e a de Uso da CPU, pois obtiveram os menores valores e muito próximos. Já em relação ao boxplot da Figura 29, a estratégia que teve o melhor desempenho em relação ao tempo foi a estratégia Round-Robin, porém teve a maior quantidade de outliers. A simetria e dispersão entre as estratégias Round-Robin, Uso da CPU e Latência tiveram praticamente os mesmos valores. A estratégia que obteve o pior desempenho foi a Random, pois além de ter um tempo maior na média, obteve um dispersão maior.

5 CONCLUSÃO

A indústria 4.0 é uma realidade que irá fazer parte de nossas vidas em um futuro muito próximo e para isso é essencial que tenhamos interoperabilidade e interação vertical de sistemas e dispositivos, com uma arquitetura distribuída e modular. Neste contexto este trabalho demonstrou que a Arquitetura Orientada a Microsserviços atende a esses quesitos essenciais para a indústria 4.0.

Foi apresentado diversos conceitos de sistemas que utilizam arquiteturas orientadas a serviço (SOA e MOA), demonstrando através de aplicações sua importância para a estrutura da Indústria 4.0. Esse tipo de arquitetura está se consolidando no setor industrial, gerando interesses em pesquisas e desenvolvimento, com o foco no desenvolvimento de ferramentas promissoras que deem suporte ao desenvolvimento da indústria 4.0.

Este trabalho apresentou a utilização e desenvolvimento de microsserviços como DAQ e *PID4.0*, além das aplicações de controle e monitoramento com orquestração dos serviços. O microsserviço DAQ faz uma operação de uma remota I/O em rede, tendo como vantagem a possibilidade de ser programável podendo ser adaptado e customizado para diversos tipos de aplicação, além de permitir uma réplica o que geraria redundância, tornando o sistema resiliente a falhas. O microsserviço de controle *PID4.0* , foco de estudo deste trabalho, flexibiliza o desenvolvimento de aplicações através do algoritmo de controle que permite ser adequado a necessidade da aplicação.

A motivação deste trabalho foi o desenvolvimento do microsserviço de controle *PID4.0* com o intuito de permitir a redundância deste tipo de controle, gerando uma grande economia de controladores físicos na planta e outros dispositivos, além de diminuir a complexidade do sistema de redundância. O microsserviço de controle flexibiliza e otimiza o desenvolvimento de algoritmo de controle conforme a necessidade do projeto, podendo ser adaptado com pequenas alterações. A malha de processo de pressão se mostrou adequada para o controle *PID4.0* redundante via rede, assim como as demais malhas disponíveis na planta piloto.

Com os experimentos realizados na planta piloto, conseguimos comprovar a efetividade da aplicação e validação do controlador redundante como um microsserviço. Os resultados apresentados demonstram que o conceito de controlador *PID4.0* redundante desenvolvido também pode ser usado para implementação de outros tipos de controladores e processos. Para este trabalho, utilizamos até 3 serviços de controle redundantes funcionando ao mesmo tempo em *Hardwares* diferentes, podendo esse ser de maior número ou menor dependendo da aplicação.

Outra contribuição importante foi a utilização das ferramentas de balanceamento de rede disponíveis no *framework Moleculer*, o que permite uma personalização do projeto de automação, permitindo uma configuração conforme a necessidade do projeto. Além disso com a análise

de desempenho e de tempo dos controladores foi possível comprovar que o PID4.0 mantém as mesmas características do PIDPlus, porém com uma vantagem, pode ser utilizado de forma redundante.

REFERÊNCIAS

AZARMIPOUR, M. et al. Plc 4.0: A control system for industry 4.0. **IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society**, 2019. DOI: 10.1109/IECON.2019.8927026. Disponível em: <https://ieeexplore.ieee.org/document/8927026>.

BANGEMANN, T. et al. State of the art in industrial automation. **Industrial cloud-based cyber-physical systems**, p. 23–47, 2014. DOI: 10.1007/978-3-319-05624-1_2.

BASSI, L. Industry 4.0: hope, hype or revolution? **2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI)**, 2017. DOI: 10.1109/RTSI.2017.8065927. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8065927>.

BELTRáN, J. V.; IGLESIAS, V. G.; FERNáNDEZ, D. R. Enabling distributed manufacturing resources through soa: The rest approach. **Robotics and Computer–Integrated Manufacturing**, v. 46, p. 156–165, 2016. Disponível em: <https://doi.org/10.1016/j.rcim.2016.09.007>.

BELTRáN, J. V. E.; IGLESIAS, V. G.; FERNNDEZ, D. R. Enabling distributed manufacturing resources through SOA: The REST approach. **Robotics and Computer-Integrated Manufacturing - Elsevier**, v. 46, p. 156–165, 2017. Disponível em: <https://doi.org/10.1016/j.rcim.2016.09.007>.

BIGHETI, J. A. **Arquitetura de Automação e Controle Orientada a Microserviços para a Indústria 4.0**. Tese (Doutorado) — Universidade Estadual Paulista, Sorocaba, São Paulo - Brasil, 2020.

BIGHETI, J. A.; FERNANDES, M. M.; GODOY, E. P. Control as a Service: A Microservice Approach to Industry 4.0. **2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0&IoT)**, 2019. DOI: 10.1109/METROI4.2019.8792918. Disponível em: <https://ieeexplore.ieee.org/document/8792918>.

BLANCHET, M. et al. **Industry 4.0 The new industrial revolution**. 2014. Disponível em: <http://www.iberglobal.com/files/Roland_Berger_Industry.pdf>. Acesso em: Abril, 2022.

BORANGIU, T. et al. Digital transformation of manufacturing through cloud services and resource virtualization. **Computers in Industry**, v. 108, p. 150–162, 2019. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0166361519300107>.

BUTZIN, B.; GOLATOWSKI, F.; TIMMERMANN, D. Microservices Approach for the Internet of Things. **IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)**, 2016. DOI: 10.1109/ETFA.2016.7733707. Disponível em: <https://ieeexplore.ieee.org/document/7733707>.

CARLSSON, O. et al. Migration of industrial process control systems to service-oriented architectures. **International Journal of Computer Integrated Manufacturing**, v. 31, n. 2, p. 156–165, 2018. Disponível em: <https://doi.org/10.1080/0951192X.2017.1392615>.

COLOMBO, A. W. et al. Industrial Cyberphysical Systems: A Backbone of the Fourth Industrial Revolution. **IEEE Industrial Electronics Magazine**, 2017. DOI: 10.1109/MIE.2017.2648857. Disponível em: <https://ieeexplore.ieee.org/document/7883993>.

DELSING, J. Local Cloud Internet of Things Automation. **IEEE INDUSTRIAL ELECTRONICS MAGAZINE**, v. 11, p. 8–21, 2017. DOI: 10.1109/MIE.2017.2759342. Disponível em: <https://ieeexplore.ieee.org/document/8241149>.

DUSTDAR, S.; PAPAZOGLOU, M. P. Services and Service Composition – An introduction. **Information Technology**, v. 50, n. 2, p. 86–92, 2008.

FERNANDES, M. M. et al. Controlador Lógico Programável como um Serviço: Uma Proposta para a Indústria 4.0. **Sociedade Brasileira de Automática**, v. 2, n. 1, 2020. Disponível em: <https://doi.org/10.48011/asba.v2i1.1740>.

FERNANDES, M. M. et al. Industrial Automation as a Service: A New Application to Industry 4.0. **IEEE LATIN AMERICA TRANSACTIONS**, v. 19, n. 12, p. 2046 – 2053, 2021. DOI: 10.1109/TLA.2021.9480146. Disponível em: <https://ieeexplore.ieee.org/document/9480146>.

FOWLER, M.; LEWIS, J. **Microservices a definition of this new architectural term**. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: Maio, 2022.

GIVEHCHI, O. et al. Control-as-a-Service from the Cloud: A Case Study for using Virtualized PLCs. **2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)**, 2014. DOI: 10.1109/WFCS.2014.6837587. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6837587>.

HEGAZY, T.; HEFFEDA, M. Industrial Automation as a Cloud Service. **IEEE Transactions on Parallel and Distributed Systems**, v. 26, n. 10, p. 2750 – 2763, 2015. DOI: 10.1109/TPDS.2014.2359894. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6908023>.

JAMMES, F. et al. Promising Technologies for SOA-Based Industrial Automation Systems. **Industrial Cloud-Based Cyber-Physical Systems**, p. 89–109, 2014. DOI: 10.1007/978-3-319-05624-1_4. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-319-05624-1_4>.

JAZDI, N. Cyber Physical Systems in the Context of Industry 4.0. **2014 IEEE International Conference on Automation, Quality and Testing, Robotics**, 2014. DOI: 10.1109/AQTR.2014.6857843. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6857843>.

KAGERMANN, H.; WAHLSTER, W.; HELBIG, J. **Recommendations for implementing the strategic initiative INDUSTRIE 4.0**. Frankfurt: Acatech, 2013.

LI, Y.; XUE, J.; GAI, Q. Design for input and output card of triple redundant control system. **Symposium on Computer Applications and Communications**, 2014. DOI: 10.1109/SCAC.2014.28. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6913176>.

LIU, G. et al. EPICS DRIVER FOR PHOENIX CONTACT REDUNDANT PLC. **Proceedings of IPAC2013**, 2013. Disponível em: <https://accelconf.web.cern.ch/ipac2013/papers/thpea016.pdf>.

LU, Y. Industry 4.0: A survey on technologies, applications and open research issues. **Journal of Industrial Information Integration**, v. 6, p. 1–10, 2017. Disponível em: <https://doi.org/10.1016/j.jii.2017.04.005>.

MELLADO, J.; NúñEZ, F. Design of an IoT-PLC: A containerized programmable logical controller for the industry 4.0. **Journal of Industrial Information Integration**, v. 25, 2022. Disponível em: <https://doi.org/10.1016/j.jii.2021.100250>.

MOLECULER. **Fast & powerful microservices framework for Node.js**. 2021. Disponível em: <https://moleculer.services/>. Acesso em: Abril, 2022.

MOREIRA, P. F. M.; BEDER, D. M. Desenvolvimento de Aplicações e Micro Serviços: Um estudo de caso. **Revista T.I.S.**, v. 4, p. 209–215, 2015. Disponível em: <http://revistatis.dc.ufscar.br/index.php/revista/article/view/364/127>.

MUBEEN, S. et al. Delay Mitigation in Offloaded Cloud Controllers in Industrial IoT. **IEEE Access**, v. 5, p. 4418 – 4430, 2017. DOI: 10.1109/ACCESS.2017.2682499. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7879156>.

NEWMAN, S. **Building Microservices**. Sebastopol, CA - Canada: O'Reilly Media, 2021.

OLLINGER, L. et al. SOA-PLC – Dynamic Generation and Deployment of Web Services on a Programmable Logic Controller. **The International Federation of Automatic Control**, v. 47, n. 3, p. 2622–2627, 2014. Disponível em: <https://doi.org/10.3182/20140824-6-ZA-1003.02189>.

ORDANINI, A.; PASINI, P. Service co-production and value co-creation: The case for a service-oriented architecture (SOA). **European Management Journal**, v. 26, n. 5, p. 289–297, 2008. Disponível em: <https://doi.org/10.1016/j.emj.2008.04.005>.

PONTAROLLI, R. P. **Composição de Serviços e Mecanismos de Segurança para Arquiteturas Orientadas a Microsserviços na Indústria 4.0**. Dissertação (Mestrado) — Universidade Estadual Paulista, Sorocaba, São Paulo - Brasil, 2020.

PONTAROLLI, R. P. et al. Planta Piloto 4.0: Uma Abordagem para a Automação e Controle de Processos Orientada a Serviços. **Congresso Brasileiro de Automática - CBA**, v. 2, n. 1, 2020. Disponível em: <https://doi.org/10.48011/asba.v2i1.1741>.

RICHARDS, M. **Microservices vs. service-oriented architecture**. 2016. Disponível em: <https://www.oreilly.com/radar/microservices-vs-service-oriented-architecture/>. Acesso em: Abril, 2022.

SAUTER, T. et al. The Evolution of Factory and Building Automation. **IEEE Industrial Electronics Magazine**, v. 5, n. 3, p. 35–48, 2011. DOI: 10.1109/mie.2011.942175. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6042563>.

SHENG, Q. Z. et al. Web services composition: A decade's overview. **Information Sciences**, v. 280, p. 218–238, 2014. Disponível em: <https://doi.org/10.1016/j.ins.2014.04.054>.

SISINNI, E. et al. Industrial Internet of Things: Challenges, Opportunities, and Directions. **IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS**, v. 14, n. 11, p. 4724 – 4734, 2018. DOI: 10.1109/TII.2018.2852491. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8401919>.

SONG, J. et al. Improving PID Control with Unreliable Communications. **ISA EXPO 2006**, 2006.

THEORIN, A.; HAGSUND, J.; JOHNSSON, C. Service Orchestration with OPC UA in a Graphical Control Language. **IEEE Emerging Technology and Factory Automation**, p. 1–6, 2014. DOI: 10.1109/ETFA.2014.7005351. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7005351>.

WOLLSCHLAEGER, M.; SAUTER, T.; JASPERNEITE, J. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. **Industrial Electronics Magazine**, v. 11, n. 1, p. 17–27, 2017. DOI: 10.1109/MIE.2017.2649104. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7883994>.

XIAO, Z.; WIJEGUNARATNE, I.; QIANG, X. Reflections on SOA and Microservices. **International Conference on Enterprise Systems**, p. 60–67, 2016. DOI: 10.1109/ES.2016.14. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7880473>.

ZHAO, Y.; LIU, F. The implementation of a dual-redundant control system. **Control Engineering Practice**, v. 12, n. 4, p. 445–453, 2004. Disponível em: <https://doi.org/10.1016/S0967-0661(03)00118-7>.