MSOFEC: Orquestração Multicamada Névoa-Nuvem para Colocação de Serviços em Internet das Coisas Industrial

Denis Contini[1], **Edmundo R. M. Madeira**[1] **e Luiz F. Bittencourt**[1]

[1]Instituto de Computação (UNICAMP), Campinas – São Paulo – Brasil

denis.contini@students.ic.unicamp.br

{edmundo,bit}@ic.unicamp.br

Abstract. *With the emergence of the Fourth Industrial Revolution, several factories began to adopt a production model that, in addition to being automated, operates with the collection and analysis of data through various sensors and applications in order to improve efficiency and productivity. As a result, the amount of data collected and transmitted has increased considerably. In this context, this work proposes a service and data orchestration mechanism based on the periodic analysis of network metrics and processing resources so that the application obtains higher levels of performance and reliability in creating the infrastructure and directing the data in a scenario fog-cloud multilayer. The initial results demonstrated that it is possible to reduce latency and improve the utilization of processing resources, maintaining good levels of availability and reliability for the application.*

Resumo. *Com o surgimento da Quarta Revolução Industrial, diversas fábricas passaram a adotar um modelo de produção que, além de automatizado, opera com a coleta e análise dos dados por meio de diversos sensores e aplicações a fim de melhorar a eficiência e produtividade. Com isso, a quantidade de dados coletados e transmitidos aumentou consideravelmente. Nesse contexto, este trabalho propõe um mecanismo de orquestração de serviços e dados baseado na análise periódica das métricas de rede e recursos de processamento de modo que a aplicação obtenha níveis maiores de desempenho e confiabilidade na criação da infraestrutura e no direcionamento dos dados em um cenário multicamada névoa-nuvem. Os resultados iniciais demonstraram ser possível diminuir a latência e melhorar a utilização de recursos de processamento, mantendo-se bons níveis de disponibilidade e confiabilidade para a aplicação.*

1. Introdução

O conceito "Indústria 4.0" surgiu em 2011 como uma iniciativa do governo alemão em melhorar a eficiência nas indústrias de manufatura. Esse paradigma tem por foco a coleta e troca de informações durante todo o ciclo de vida de um produto [Khan et al. 2020].

Assim como ocorre em aplicações de Internet das Coisas, o conceito de Internet das Coisas Industrial (*Industrial Internet of Things* - IIoT) apresenta a ideia de que os diversos sensores, atuadores e aplicações da indústria estejam interconectados e compartilhem dados não apenas entre si, mas também com sistemas remotos que analisam as informações e auxiliam na tomada de decisões [Xu et al. 2014].

Entretanto, há um ponto em que os conceitos podem divergir: na quantidade e na periodicidade em que os dados são gerados e transmitidos pela rede. Empresas de grande porte, geralmente, possuem um número elevado de sensores e atuadores em suas linhas de produção transmitindo e/ou consumindo enormes quantidades de dados em curtos períodos de tempo com o intuito de analisar e melhorar os processos de fabricação em geral. Aplicações IoT, por outro lado, podem ter a coleta e a transmissão dos dados menos periódicas e em menores quantidades, dependendo do cenário [Yang et al. 2016].

Diante do exposto, fica evidente a necessidade de uma infraestrutura que ofereça suporte e recursos suficientes para que os requisitos sejam atendidos de forma satisfatória em ambientes com dispositivos e aplicações heterogêneos. Essa infraestrutura também precisa apresentar níveis maiores de flexibilidade, escalabilidade, confiabilidade e dinamicidade na localização dos dispositivos para onde os dados serão encaminhados [Yang et al. 2016, Khan et al. 2020], levando-se em conta cenários onde métricas de rede como a latência, largura de banda e custos de comunicação, por exemplo, apresentem-se como fatores críticos [Khan et al. 2020, Escamilla-Ambrosio et al. 2018].

Diferente dos trabalhos que buscam resolver questões de alocação de serviços trazendo-os para camadas mais próximas da borda da rede, dessa forma clusterizando-os horizontalmente em uma única camada, esse trabalho propõe dois mecanismos orquestradores. O primeiro tem por objetivo a disponibilização da infraestrutura e distribuição de serviços entre as múltiplas camadas (borda, névoa e nuvem). O segundo, por sua vez, busca distribuir os dados provindos dos sensores e das aplicações da borda da rede na camada que melhor possa atender aos requisitos necessários para o bom funcionamento da aplicação, aproveitando os benefícios e vantagens que cada uma pode proporcionar.

Sendo assim, os mecanismos propostos devem efetuar o monitoramento periódico das métricas de rede e dos recursos de processamento disponíveis e, com base nas informações obtidas, distribuir os serviços e dados nas camadas que melhor possam atender a aplicação, operando, dessa forma, como um complemento aos orquestradores convencionais, permitindo uma distribuição melhor coordenada através de múltiplas camadas (borda, névoa e nuvem).

Para validar a proposta, foram realizados experimentos reais onde dados eram transmitidos e consumidos em curtos períodos de tempo, buscando-se replicar os ambientes das fábricas inteligentes onde grandes quantidades de dados trafegam entre os dispositivos a todo momento.

O restante do trabalho está estruturado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados à orquestração de serviços; na Seção 3, é apresentada a arquitetura dos mecanismos propostos; na Seção 4, são especificados o cenário proposto para rodar os experimentos, a metodologia utilizada, bem como a análise dos resultados obtidos. Por fim, a Seção 5 apresenta a conclusão do trabalho.

2. Trabalhos Relacionados

Nos últimos anos, tem sido comum encontrar *cases* e aplicações IoT tendo seus dados processados e/ou armazenados não apenas em servidores na nuvem, mas também em camadas mais próximas da própria aplicação, conhecidas como *Edge* (Borda) e *Fog* (Névoa), como pode ser observado nos trabalhos de [Bittencourt et al. 2017, Contini et al. 2020].

Com essa "migração" das aplicações e dispositivos para camadas mais próximas de onde os dados são gerados e consumidos, respostas mais rápidas e redução de custos podem ser alcançados, além da utilização eficiente da largura de banda.

Apesar dos benefícios da utilização de infraestruturas menos robustas e mais próximas da borda da rede, existem casos onde o grande volume de dados gerados pelas aplicações necessitam de maior poder de processamento ou armazenamento, tornando inviável abandonar completamente os recursos providos pelos servidores em nuvem.

Diante dos dilemas acima descritos, pesquisadores têm proposto diversas abordagens de modo que os benefícios de cada cenário possam ser melhor explorados, conforme as necessidades e características específicas das aplicações.

Em busca de uma plataforma de orquestração que possa atender de maneira eficiente a criação de *clusters* para o processamento de dados de aplicações IoT, [Fayos-Jordan et al. 2020][1] apresentam um estudo baseado na realização de testes de desempenho dos principais orquestradores disponíveis executando em dispositivos com poder computacional limitado (*Small Board Computers*). Após as análises, os autores concluíram que, embora o *Kubernetes* apresente um conjunto de recursos que pode ser útil em computadores de alto desempenho, o *Swarm* provê a utilização mais eficiente dos recursos de processamento em dispositivos com baixo poder computacional.

Em seu trabalho, [Bellavista and Zanni 2017][2] propuseram uma arquitetura de referência baseada em um *framework* descentralizado e conteinerizado em névoa, onde um *cluster* de *gateways* divide a carga de trabalho, atuando como um *midleware* para a aplicação de modo geral. Ao validar a proposta, concluíram que a abordagem gerou melhorias e facilidades na escalabilidade, além de permitir maior flexibilidade e redução de custos ao operar em conjunto com dispositivos de poder computacional limitado.

Em [Bittencourt et al. 2017][3], os autores buscam uma possível solução para o problema de alocação de serviços entre camadas hierárquicas levando em conta situações de mobilidade dos usuários por meio de simulações. Para isso são propostas estratégias de colocação de serviços, onde os dispositivos de processamento em névoa buscam atender às aplicações, submetendo os dados para a nuvem apenas em casos onde o usuário muda a sua localização ao longo do tempo e o serviço precisa ser transferido para um novo dispositivo ou, ainda, em situações onde o dispositivo não disponha de recursos suficientes para atender a tarefa.

Diante das lacunas em aberto nos mecanismos de orquestração de serviços existentes, em [Hoque et al. 2017][4], os autores propõem um *framework* orquestrador. Baseado no *engine* do *Swarm*, o *framework* sugere a manipulação das políticas de orquestração através da API disponível. O trabalho não cita métricas específicas a serem monitoradas e manipuladas, entretanto propõe distribuição dos serviços entre dispositivos alocados na camada de névoa, com o intuito de diminuir o tempo nas respostas.

Embora os trabalhos acima relacionados proponham possíveis soluções para o paradigma de orquestração de serviços e infraestruturas para IoT, a maioria aborda a distribuição dos mesmos em uma ou duas camadas, permitindo uma clusterização horizontal, onde apenas dispositivos no mesmo nível acabam por compartilhar seus recursos.

Na Tabela 1, é possível verificar as principais diferenças entre os trabalhos anali-

Tabela 1. Características dos trabalhos analisados

Trabalhos	Orquestrador	Camada	Experimentos	Métricas Orquestração
[1]	Swarm / Kubernetes Padrão	Névoa	Disp. Reais; Amb. Controlado; Topologia Estática	Disponibilidade nó; Recursos de Hw e Sw; Políticas; Restrições
[2]	Swarm Padrão	Névoa	Disp. Reais; Amb. Controlado; Topologia Estática	Disponibilidade nó; Recursos de Hw e Sw; Políticas; Restrições
[3]	Algoritmo Orquestrador	Névoa/ Nuvem	Simulações	Disponibilidade nó; Recursos de CPU; Latência; Prioridade; Mobilidade
[4]	Swarm Modificado	Névoa	Framework Conceitual	Disponibilidade nó; Recursos de Hw e Sw; Políticas; Restrições e outros não especificados
MSOFEC	Swarm + Mecanismo Orquestrador	Multi-camadas	Disp. Reais; Amb. Rede Real; Topologia Dinâmica	Disponibilidade nó; Recursos de Hw e Sw; Políticas; Restrições; Métricas de Rede

sados e o trabalho proposto. Levando-se em conta que cada camada possui suas vantagens e desvantagens variando conforme o cenário onde são avaliadas, mostra-se adequada uma abordagem que leve em consideração uma clusterização vertical, onde dispositivos heterogêneos em camadas distintas possam compartilhar recursos e melhorar o desempenho das aplicações. Para isso, o mecanismo orquestrador deve analisar as métricas relevantes para classificação e agrupamento das camadas e, então, definir onde os serviços e dados podem ser alocados, conforme os requisitos exigidos pela aplicação.

Através da clusterização vertical, os recursos de mais de uma camada podem ser melhor aproveitados, a infraestrutura pode atingir níveis maiores de confiabilidade, escalabilidade e dinamicidade, além de, em alguns casos, reduzir custos desnecessários e a utilização exagerada da largura de banda disponível com tráfego excessivo entre a borda e a nuvem, conforme ocorre em trabalhos que buscam a orquestração dos serviços em apenas uma camada.

3. Arquitetura de Orquestração Proposta

Com o intuito de aproveitar as vantagens e os recursos de cada uma das camadas (borda, névoa e nuvem), este trabalho propõe o desenvolvimento de um mecanismo orquestrador de serviços e dados em múltiplas camadas chamado **MSOFEC** - ***M**ultilayer **S**ervice **O**rchestrator in **F**og, **E**dge, and **C**loud*.

O MSOFEC visa a orquestração de serviços e dados através da análise de métricas de rede e recursos computacionais de maneira que dispositivos de processamento sejam agrupados em camadas distintas e atendam diferentes requisitos e níveis de prioridade (LoP - *Levels of Priority*) das aplicações e/ou grupos de sensores e atuadores encontrados nas fábricas inteligentes.

Na Figura 1(a), são apresentados os módulos essenciais para o funcionamento do Mecanismo Orquestrador de Serviços (MOS), bem como as dependências entre eles. O módulo de monitoramento desempenha a função de verificar as métricas relacionadas à rede (latência, largura de banda, perda de pacotes, *hops*) e aos recursos de processamento (CPU e memória RAM) através de medições em períodos de tempo predefinidos, de modo que a topologia possa se readequar frente a possíveis alterações ao longo de sua execução. Uma vez executadas as medições pelo módulo de monitoramento, o módulo gerenciador

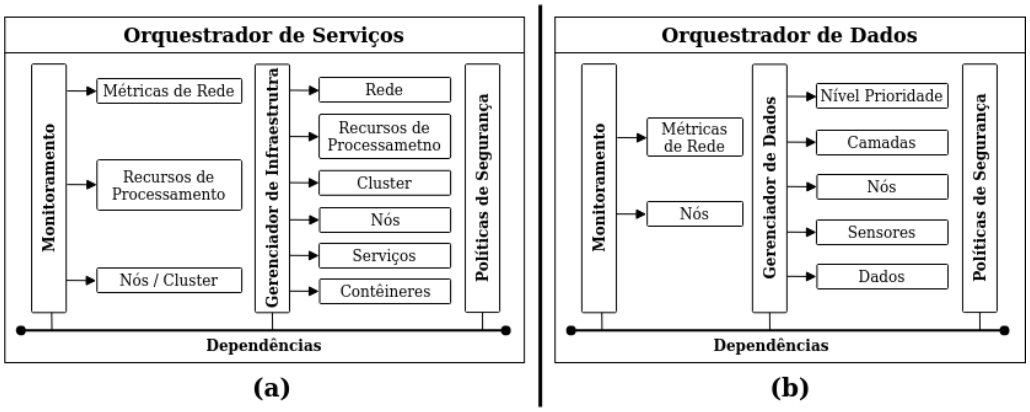

Figura 1. (a) Orquestrador de Serviços e (b)Orquestrador de Dados

de infraestrutura inicia o processo de criação ou reestruturação dos grupos e camadas conforme os valores obtidos para, então, iniciar a distribuição dos serviços entre os nós.

A Figura 1(b), por sua vez, apresenta os módulos presentes no Mecanismo Orquestrador de Dados (MOD). Uma vez que a infraestrutura já se encontra operacional, o módulo de monitoramento do mecanismo orquestrador de dados verifica as métricas de rede entre o *gateway* da aplicação IIoT, na borda da rede, e demais nós para, então, definir para qual camada os dados serão enviados, conforme os níveis de prioridade de cada grupo de sensores. Feito isso, o módulo gerenciador de dados se encarrega de distribuir os dados através das camadas, baseando-se nas leituras periódicas obtidas pelo módulo de monitoramento.

Durante os processos de monitoramento e gerenciamento de ambos os mecanismos, o módulo políticas de segurança é o responsável pela segurança dos dados e serviços entre as camadas e os nós, tornando a comunicação mais segura.

3.1. Requisitos da Infraestrutura

Ao se trabalhar com ambientes multicamadas, alguns desafios podem ser mencionados, entre eles: heterogeneidade de dispositivos e recursos, diferentes larguras de banda, políticas de segurança das redes onde os nós estão inseridos, além de outras questões que acabam por dificultar a comunicação entre os nós ao se formar um *cluster* para compartilhamento de recursos.

Para lidar com esses desafios, a utilização de contêineres e orquestradores se apresenta como uma boa alternativa, uma vez que independem do *hardware* e do sistema operacional dos nós onde estão em execução, podendo, até, funcionar em conjunto com máquinas virtuais [Docker 2021]. Diferente dos tradicionais servidores físicos ou das máquinas virtuais, contêineres, de modo geral, podem ser considerados um tipo de virtualização leve. Neles, é possível incluir serviços, dados e códigos de forma isolada do sistema operacional do nó onde estão em execução, reduzindo a utilização de espaço em disco e o consumo dos recursos de processamento [Fayos-Jordan et al. 2020, Do Espírito Santo et al. 2019, Hoque et al. 2017, Docker 2021].

Na maioria das vezes, podem ser manipulados e gerenciados através de Interfaces de Programação de Aplicativos (APIs) ou Interface de Linha de Comando (CLI - *Com-*

mand Line Interface) [Kubernetes 2021, Docker 2021].

Conforme a quantidade de contêineres aumenta, são necessários mecanismos de orquestração para que os mesmos sejam distribuídos e administrados de forma eficiente entre os nós do *cluster*. Embora possuam objetivos semelhantes, cada orquestrador apresenta seus mecanismos particulares para descoberta de nós, políticas de acesso e gerenciamento de toda a infraestrutura, incluindo o gerenciamento da comunicação entre os nós, contêineres e serviços. Isso faz com que cada um tenha suas vantagens e desvantagens, podendo apresentar melhor ou pior performance em determinados cenários [Fayos-Jordan et al. 2020].

Para auxiliar no processo de distribuição dos serviços, optou-se pela ferramenta de orquestração *Docker Swarm* devido à sua simplicidade e ao baixo consumo dos recursos de processamento quando em execução em dispositivos com baixo poder computacional [Fayos-Jordan et al. 2020, Hoque et al. 2017, Bellavista and Zanni 2017].

3.2. Orquestração de Serviços

Para que a topologia multicamadas seja criada e os serviços possam ser atribuídos à camada que melhor atenda aos seus requisitos, o Mecanismo Orquestrador de Serviços deve, através do módulo de monitoramento, iniciar a coleta das informações dos recursos de processamento e das métricas de rede entre os nós disponíveis no *cluster*. Entre as informações a serem monitoradas, é possível mencionar: a distância lógica (*hops*), a latência média, o percentual de pacotes perdidos na comunicação, a largura de banda útil (*Download* e *Upload*), a quantidade total de memória RAM e dos núcleos do processador de cada nó.

Com base nas informações obtidas, o mecanismo orquestrador deve agrupar os nós que apresentarem métricas semelhantes, conforme requisitos estabelecidos e parametrizados previamente. Dessa forma, dispositivos com baixo poder computacional mas que apresentem baixos valores de latência e estejam próximos uns dos outros, por meio da verificação da distância lógica, podem ser agrupados na camada de borda, assim como dispositivos com valores médios de processamento e latência ou altos recursos de processamento, que apresentem altos valores de latência e distantes da borda podem ser agrupados nas camadas de névoa e nuvem, respectivamente.

Na Figura 2, é possível verificar, em maiores detalhes, os fluxos da troca de informações entre os módulos do Mecanismo Orquestrador de Serviços, bem como a sua interação com a API da ferramenta de orquestração de contêineres. Com base nas informações coletadas pelo módulo de monitoramento, que possui a visão geral dos nós da topologia, o módulo gerenciador de infraestrutura inicia o processo de criação e manipulação das políticas de distribuição dos serviços no nó gerenciador do *cluster* (*master*) localizado na borda da rede que, por sua vez, é o responsável pela criação do objeto serviço que contém as tarefas e configurações necessárias para que o serviço seja distribuído entre os nós das múltiplas camadas e se torne disponível para a aplicação.

É importante destacar que as medições das métricas de rede e dos recursos de processamento executadas pelo módulo de monitoramento devem ser feitas periodicamente, através de intervalos de tempo predefinidos, para que a topologia possa assumir características dinâmicas e se reorganizar em casos de alterações nas informações monitoradas.

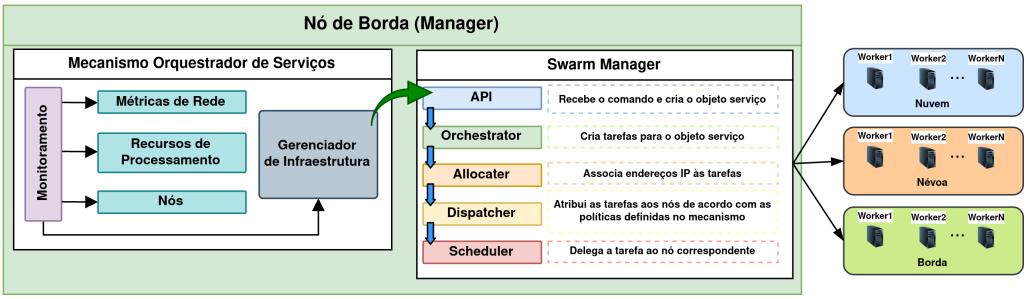

Figura 2. Componentes da arquitetura do Mecanismo Orquestrador de Serviços

3.3. Orquestração de Dados

Antes de começar a transmitir os dados provenientes dos sensores da aplicação de forma aleatória entre os dispositivos de processamento e/ou armazenamento, um segundo mecanismo orquestrador, dessa vez de dados, inicia o processo de verificação das métricas de rede entre o *gateway* da aplicação, geralmente localizado na borda da rede, e os nós disponibilizados previamente pelo Mecanismo Orquestrador de Serviços.

Durante o processo, as informações de latência média, distância lógica e a taxa de perda de pacotes entre os nós são analisadas. Diferente do Mecanismo Orquestrador de Serviços, o Mecanismo Orquestrador de Dados não analisa a largura de banda útil e os recursos de processamento, uma vez que já foram verificados e agrupados durante o processo de orquestração de serviços.

Após a coleta e análise das métricas, o Mecanismo Orquestrador de Dados poderá classificar e agrupar sensores em diferentes níveis de prioridade ou pré-requisitos, conforme as necessidades da aplicação. Através dessa classificação, dados que necessitem de valores reduzidos de latência e baixo poder computacional, por exemplo, podem ser direcionados para os dispositivos em borda.

Nos casos onde os dados necessitam de maior poder computacional e possuem uma tolerância levemente maior aos atrasos na entrega dos pacotes, os mesmos podem ser redirecionados para a névoa. Em casos mais extremos, onde a baixa latência não é o principal requisito a ser levado em conta, mas sim o alto poder de processamento requerido pela aplicação e os recursos de borda e névoa não atendam às necessidades, os dados possam ser redirecionados para a nuvem.

Esse encaminhamento dinâmico busca tornar o processamento e armazenamento dos dados e informações mais eficiente, flexível e escalável para a aplicação, mantendo características simples e fáceis para o seu gerenciamento e funcionamento. Além disso, buscam melhorias na utilização dos enlaces de comunicação, reduzindo o volume de dados que trafegam entre a borda e a nuvem, podendo, também, gerar redução nos custos com a utilização de recursos em nuvem.

4. Resultados Experimentais

Uma vez definidos os componentes e a arquitetura dos mecanismos orquestradores, surge a necessidade de um cenário onde os mesmos possam ser testados e avaliados. Nesta seção, são apresentados detalhes do cenário para realização dos experimentos, bem como os recursos computacionais utilizados e a metodologia.

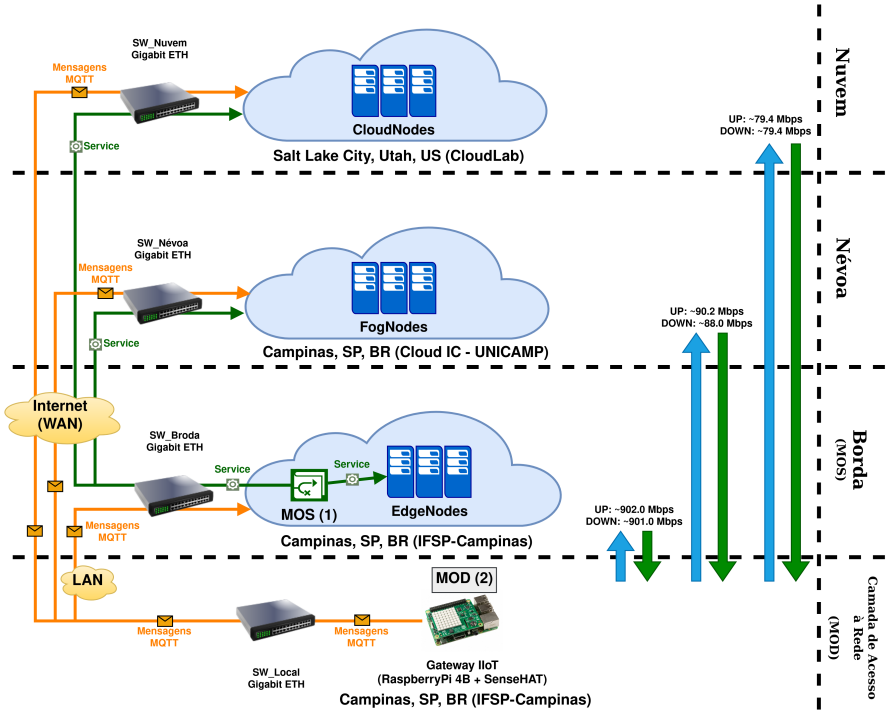

Figura 3. Topologia do cenário de testes

4.1. Cenário de Testes

Ao invés de utilizar simuladores ou emuladores de rede, neste trabalho, optou-se pela implementação de um cenário real, onde os valores obtidos pelo módulo de monitoramento fossem semelhantes aos encontrados em aplicações reais. Na Figura 3, é possível verificar a topologia utilizada, bem como a distribuição da infraestrutura em um cenário multicamadas onde, em um primeiro momento, ocorre a orquestração dos serviços (1) para, em seguida, iniciar a orquestração dos dados (2).

Para a execução dos experimentos, foram utilizados três dispositivos com configurações diferentes, alocados em redes e em camadas distintas (Figura 3). Com exceção do dispositivo de borda, onde a orquestração dos serviços se deu em um servidor *bare metal* (máquina física), os outros dois dispositivos, alocados em névoa e nuvem, respectivamente, são máquinas virtuais.

Na Tabela 2, é possível verificar as especificações do *hardware* dos nós que compõem o cenário de testes, tornando evidentes as diferenças do poder de processamento de cada um. O Sistema Operacional utilizado pelos nós onde os serviços são distribuídos foi o Ubuntu Server 18.04 64 Bits, Kernel versão 4.15.0-136-generic.

Tabela 2. Especificações dos dispositivos de processamento

Camada	Físico/ Virtual	CPU	# Cores	Freq. CPU (GHz)	RAM (GB)
Borda	Físico	Intel(R) Xeon(R) Gold 6130	2	2.10	4
Névoa	Virtual	Commom KVM Processor	4	2.10	8
Nuvem	Virtual	Intel(R) Pentium(R) CPU G850	64	2.90	192
RPi 4B	Físico	ARM Cortex-A72	4	1.50	4

Além das especificações de *hardware* distintas, os nós foram alocados distantes geograficamente uns dos outros, de forma que o mecanismo orquestrador pudesse classificá-los conforme a análise das métricas de rede que, propositalmente, deveriam divergir entre si, principalmente na distância lógica (quantidade de saltos) e nas latências.

Embora existam outros mecanismos para orquestração de contêineres, neste trabalho, optou-se pela utilização do *Docker Swarm* devido à sua simplicidade e o consumo reduzido de recursos de processamento quando comparado às outras ferramentas, conforme pode ser visto em [Fayos-Jordan et al. 2020, Hoque et al. 2017].

Para simular o envio de dados pelos sensores na borda da rede, foi utilizado um Raspberry Pi 4B, cujas configurações de *hardware* podem ser vistas na Tabela 2, tendo suas funções gerenciadas pelo Sistema Operacional Raspbian GNU/Linux 10 32 Bits.

Nele, uma aplicação desenvolvida em Python opera em conjunto com o emulador de sensores *Sense HAT Emulator* [Raspberry Pi Foundation 2021], sendo a responsável pelo envio dos dados dos sensores de temperatura, umidade e pressão, de modo a replicar um cenário de linha de produção de uma fábrica inteligente em menor escala.

O protocolo de comunicação utilizado para a transmissão dos dados foi o MQTT [MQTT 2021, Khan et al. 2020], devido à sua simplicidade, eficiência, comunicação bidirecional, escalabilidade e entrega confiável de mensagens, amplamente utilizados para transmissão dos dados em redes industriais.

Durante os experimentos, um total de 3 mil mensagens com os dados dos sensores foram transmitidas para o serviço de *Message Broker* em execução em cada uma das camadas criadas pelo Mecanismo Orquestrador de Serviços. O *Message Broker* atua como um serviço intermediário para o enfileiramento de mensagens para que as aplicações possam consumi-los instantaneamente ou posteriormente, utilizando-se do protocolo AMQP [AMQP 2021]. O tempo médio para o envio das mensagens foi de 3 segundos, atingindo uma taxa aproximada de mil mensagens por segundo para cada cenário avaliado.

Para a execução do experimento, foram definidos 5 cenários, conforme apresentado na Tabela 3. Em cada um dos cenários, foram executadas duas rodadas. Na primeira (WS - *With Subscriber*), o cenário contava com um dispositivo *Publisher*, enviando os dados dos sensores e uma aplicação *Subscriber*, consumindo instantaneamente os dados. Na segunda (WOS - *Without Subscriber*), a aplicação *Subscriber* estava ausente, fazendo com que as mensagens enviadas pelo *Publisher* ficassem armazenadas nas filas do *Message Broker*.

Tabela 3. Cenários do experimento

Cenário	Camada	# Sensores	Orquestração	# Contêineres
1	Borda	3	Nenhuma	1
2	Névoa	3	Nenhuma	1
3	Nuvem	3	Nenhuma	1
4	Multicamada	3	Swarm Padrão	3
5	Multicamada organizados em grupos	9	Mecanismo Orquestrador de Serviços e Dados	3

Os cenários 1, 2 e 3 contam com apenas um nó e um serviço por camada (borda, névoa e nuvem). Para cada um dos 3 cenários, as mensagens foram enviadas com o intuito de verificar a utilização dos recursos de processamento nos nós e no contêiner, além do

Tempo Médio Total (T_{MT}) gasto para que cada mensagem fosse enviada do sensor e consumida pela aplicação *Subscriber* (para o caso da primeira rodada - WS). Em nenhum dos 3 cenários foram utilizados recursos de clusterização ou orquestração.

No cenário 4, o mecanismo padrão para orquestração de contêineres do *Swarm* foi configurado de modo que os 3 nós se juntaram para formar um *cluster*, sem que houvesse uma divisão em camadas, operando como se fosse um conjunto de recursos horizontais, apenas.

Independente do nó para o qual os dados dos sensores forem enviados e/ou consumidos, o *Swarm* efetua um balanceamento de carga, dividindo as requisições entre os nós do *cluster* em que o serviço está em execução. Diante desse fato, foram executadas rodadas extras nos experimentos do cenário 4, onde os dados coletados refletem a média aritmética dos valores obtidos para situações onde os dados eram enviados e consumidos em um endereço IP de borda, névoa e nuvem, respectivamente.

Por fim, o cenário 5 apresenta o funcionamento dos mecanismos de orquestração de serviços e dados propostos neste trabalho formando, então, uma topologia com múltiplas camadas para onde os dados da aplicação podem ser enviados, com o auxílio do MOD, para os diversos serviços disponibilizados pelo MOS.

Quando em execução nos experimentos do cenário 5, o Mecanismo Orquestrador de Serviços efetua a verificação dos nós disponíveis no *cluster*. Durante o processo, informações sobre a distância lógica, latência média, largura de banda útil, taxa de perda de pacotes no enlace, quantidade de núcleos do processador e memória RAM total dos nós são analisadas em tempo real e, com base nas especificações do mecanismo, três grupos são criados: *Edge*, *Fog* e *Cloud*. As medições ocorrem sempre de um dispositivo alocado na camada de borda para os demais.

Embora os parâmetros pudessem ser alterados através de um arquivo de configuração, nós que apresentassem uma latência menor ou igual a 5 ms, distância lógica menor ou igual a 3 saltos, uma quantidade de Memória RAM menor ou igual a 4GB e um processador com 2 núcleos ou menos, deveriam ser classificados como nós de borda.

Dispositivos que apresentassem uma latência maior que 5 ms e menor que 80 ms, distância lógica maior que 3 e menor que 11 saltos, uma quantidade de Memória RAM acima de 4GB e abaixo de 24GB e um processador com um número de núcleos maior que 2 e menor que 24, seriam classificados como nós de névoa.

Por fim, para serem classificados como nós de nuvem, as configurações deveriam apresentar valores de latência maiores ou iguais que 80 ms, distância lógica maior ou igual que 11 saltos, memória RAM maior ou igual que 24GB e um processador com 24 núcleos ou mais.

Embora a topologia proposta se forme nas 3 camadas, os mecanismos atuam de forma dinâmica, executando medições periódicas através do módulo de monitoramento. Sendo assim, a topologia disponível pode, em alguns casos, não apresentar, necessariamente, dispositivos em todas as camadas.

4.2. Latência e utilização dos recursos

Primeiramente, buscou-se analisar os valores da latência média total do instante em que o dado era transmitido dos sensores para o *Message Broker*, até serem consumidos pela aplicação *Subscriber*, em tempo real, ou seja, os dados eram enviados e consumidos pela aplicação instantaneamente.

Em um segundo momento, foram analisadas as informações relacionadas ao consumo de processador e memória RAM nos nós e contêineres com o experimento em execução, levando-se em consideração rodadas onde os dados eram consumidos e rodadas onde não eram consumidos, ou seja, permaneciam nas filas do *Messge Broker*.

Para que fosse possível o cálculo das médias de latência (L_{MT}), contabilizadas entre o momento do envio dos dados até a sua utilização, foram utilizadas variáveis para o registro do *timestamp* do dispositivo *Publisher* (T_{Pub}) onde os sensores eram emulados.

As informações de data e hora eram registradas e enviadas no *payload* da mensagem, junto com as informações dos sensores. Ao chegarem na aplicação consumidora, os valores enviados eram subtraídos dos valores de data e hora atualizados no dispositivo onde as informações eram utilizadas (T_{Sub}), utilizando-se a fórmula $L_{MT} = T_{Sub} - T_{Pub}$. Para melhor acurácia, todos os dispositivos estavam sincronizados via NTP com o horário oficial de Brasília (GMT -3).

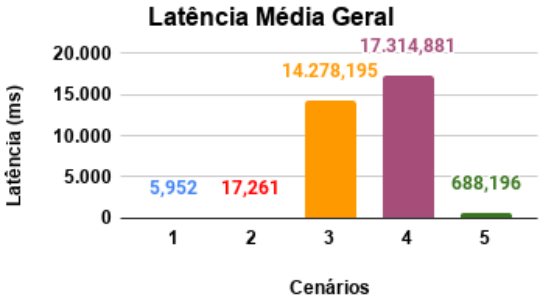

Figura 4. Latência Média Geral - Com *Subscriber*

Durante as medições do cenário 4, notou-se que, ao configurar o envio e consumo das mensagens em camadas diferentes, os valores apresentavam diferenças consideráveis. Diante dessa variação, para este cenário, foram executadas rodadas extras, onde a aplicação publicava e consumia os dados nas múltiplas camadas. Findadas as execuções das rodadas, calculou-se a média aritmética entre os valores observados, utilizando a mesma para fins de comparação.

Conforme pode ser visto na Figura 4, os resultados iniciais indicaram que os valores médios de latência ao se enviar os dados para os cenários 1 e 2 mantiveram-se abaixo dos 20 ms, comprovando as baixas latências ao se utilizar dispositivos em borda e névoa. Para os cenários 3 e 4, os valores de latência mostraram-se bastante elevados, ultrapassando a casa dos 10 segundos de tempo médio.

No cenário 5, onde são utilizados os mecanismos de orquestração de serviços e dados propostos neste trabalho, é possível observar uma latência média próxima de meio segundo, tornando evidente a redução na latência quando comparado aos cenários 3 e 4.

A seguir, na Figura 5, são apresentados os dados referentes à utilização de processador e memória RAM dos nós e contêineres com uma aplicação *Subscriber* (5a e 5b) e sem *Subscriber* (5c e 5d).

Comparando os resultados obtidos e apresentados nos gráficos das Figuras 4 e 5 em conjunto, é possível notar que, embora o cenário 1 apresente a menor taxa de latência durante o processo e envio e utilização dos dados, o consumo dos recursos de processamento é mais elevado (Fig. 5a e 5b, principalmente), devido às limitações do *hardware* utilizado. Com isso, é possível prever que, escalando-se a quantidade de mensagens transmitidas e consumidas pela aplicação, o nó poderá se sobrecarregar e apresentar um aumento na latência média.

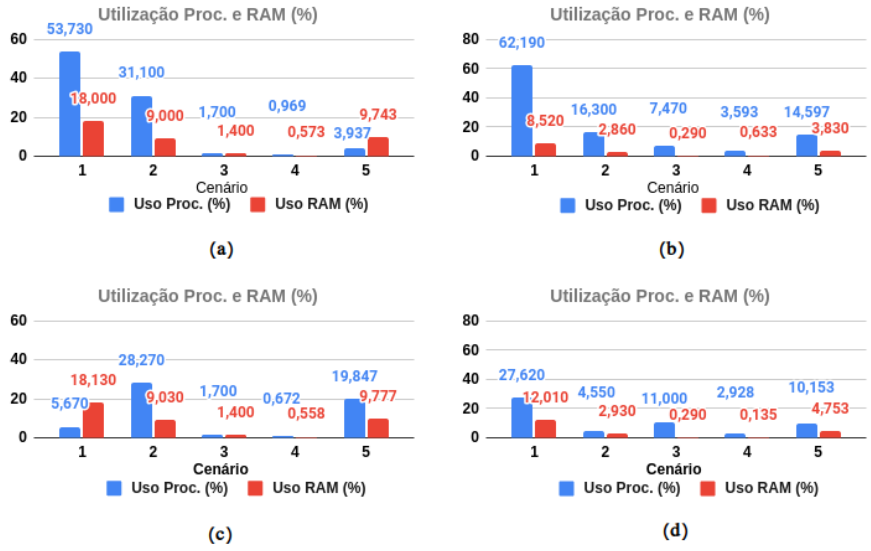

Figura 5. Média de utilização dos recursos de processamento

Já no cenário 2, é possível verificar, também, baixas latências e um consumo moderado dos recursos de *hardware*, embora maiores que o observado nos cenários 3 e 4 devido ao elevado poder de processamento dos nós de nuvem e do *cluster*. Além disso, é na névoa que se encontra o gerenciador do *cluster* em execução, o que ocasiona um leve aumento na utilização dos recursos de processamento quando os cenários 4 e 5 estão em execução.

Nos cenários 3 e 4, embora a utilização dos recursos de processamento seja bastante reduzida (quase sempre abaixo dos 10% nos experimentos realizados), os altos valores de latência observados na Figura 4 apontam um possível ponto crítico para a utilização da topologia proposta nos cenários em aplicações de IIoT que, em grande parte, apresentam-se sensíveis às altas latências.

No cenário 5, onde são aplicados os mecanismos de orquestração propostos, é possível notar uma redução considerável na latência quando comparados aos valores obtidos nos cenários 3 e 4 (95,18% e 96,03%, respectivamente), apontando ganhos significativos na utilização dos mecanismos propostos para distribuição dos serviços de dados.

Quando comparada com os resultados obtidos no cenário 1, a solução proposta apresentou uma redução de 92,67% (Fig. 5a), 76,53% (Fig. 5b) e 63,24% (Fig. 5d) na

utilização do CPU e 45,87% (Fig. 5a), 55,05% (Fig. 5b), 46,07% (Fig. 5c) e 60,42% (Fig. 5d) na utilização da memória RAM. Já na comparação com o cenário 2, a redução foi de 87,34% (Fig. 5a), 10,45% (Fig. 5b) e 29,80% (Fig. 5c) na utilização do CPU, apresentando valores aproximados quanto à utilização da memória RAM.

Além dos ganhos com a latência, ao se analisar a utilização dos recursos de processamento, é possível notar um baixo consumo devido à distribuição coordenada dos serviços e dados entre as camadas, podendo retardar ou até mesmo evitar situações onde os dispositivos de processamento fiquem sobrecarregados e deixem de atender a aplicação ou aumentem os tempos de resposta.

5. Conclusão

Em aplicações onde existem grandes quantidades de dados a serem transmitidas em curtos períodos de tempo é importante a utilização de uma topologia de rede flexível e escalável, onde os dispositivos possam compartilhar recursos e atender aos requisitos das aplicações com eficiência, flexibilidade, alta disponibilidade e confiabilidade.

Buscando atender as necessidades acima descritas, este trabalho propôs a utilização de mecanismos de orquestração de serviços e dados em múltiplas camadas baseados na leitura e análise das métricas de rede e dos recursos de processamento dos nós disponíveis. Por meio de experimentos reais, foram analisados os valores de latência média e o percentual de utilização dos recursos de processamento nos cenários propostos.

Através dos resultados iniciais, foi possível notar que o mecanismo proposto obteve reduções consideráveis nos valores de latência quando comparados aos cenários de nuvem e com um *cluster* geral, sem a diferenciação por camadas. Além disso, a utilização dos recursos de processamento manteve-se reduzida, apontando um retardo na sobrecarga dos nós, diferente do observado ao se enviar os dados todos apenas para a borda.

Como melhorias futuras, pretendemos a integração com algoritmos de aprendizado de máquina para análise de padrões e mudanças dinâmicas na topologia baseadas no histórico de informações, de modo que o direcionamento dos serviços e dados apresentem níveis maiores de precisão, além da utilização dos mecanismos propostos em cenários reais de fábricas inteligentes, com enormes quantidades de sensores e dados.

Agradecimentos

Esta pesquisa é parte do INCT da Internet do Futuro para Cidades Inteligentes, financiado por CNPq (proc. 465446/2014-0), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001 e FAPESP (procs. 14/50937-1 e 15/24485-9). Os autores agradecem, também, à Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP (#2015/24494-8) e o Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Referências

AMQP (2021). Amqp - advanced message queuing protocol. Technical report, https://amqp.org – Acessado em 15/03/2021.

Bellavista, P. and Zanni, A. (2017). Feasibility of fog computing deployment based on docker containerization over RaspberryPi. In *ACM International Conference Proceeding Series*. Association for Computing Machinery.

Bittencourt, L. F., Diaz-Montes, J., Buyya, R., Rana, O. F., and Parashar, M. (2017). Mobility-Aware Application Scheduling in Fog Computing. *IEEE Cloud Computing*, 4(2):26–35.

Contini, D., De Castro, L. F. S., Madeira, E., Rigo, S., and Bittencourt, L. F. (2020). Simulating Smart Campus Applications in Edge and Fog Computing. In *Proceedings - 2020 IEEE International Conference on Smart Computing, SMARTCOMP 2020*, pages 326–331. Institute of Electrical and Electronics Engineers Inc.

Do Espírito Santo, W., De Souza Matos, R., De Ribamar Lima Ribeiro, A., Silva, D. S., and Santos, R. (2019). Systematic Mapping on Orchestration of Container-based Applications in Fog Computing. In *15th International Conference on Network and Service Management, CNSM 2019*. Institute of Electrical and Electronics Engineers Inc.

Docker (2021). Docker overview. Technical report, https://docs.docker.com/get-started/overview – Acessado em 01/03/2021.

Escamilla-Ambrosio, P. J., Rodríguez-Mota, A., Aguirre-Anaya, E., Acosta-Bermejo, R., and Salinas-Rosales, M. (2018). Distributing computing in the internet of things: Cloud, fog and edge computing overview. *Studies in Computational Intelligence*, 731:87–115.

Fayos-Jordan, R., Felici-Castell, S., Segura-Garcia, J., Lopez-Ballester, J., and Cobos, M. (2020). Performance comparison of container orchestration platforms with low cost devices in the fog, assisting Internet of Things applications. *Journal of Network and Computer Applications*, 169:102788.

Hoque, S., Brito, M. S. D., Willner, A., Keil, O., and Magedanz, T. (2017). Towards Container Orchestration in Fog Computing Infrastructures. In *Proceedings - International Computer Software and Applications Conference*, volume 2, pages 294–299. IEEE Computer Society.

Khan, W. Z., Rehman, M. H., Zangoti, H. M., Afzal, M. K., Armi, N., and Salah, K. (2020). Industrial internet of things: Recent advances, enabling technologies and open challenges. *Computers and Electrical Engineering*, 81:106522.

Kubernetes (2021). What is kubernetes? Technical report, https://kubernetes.io/docs/concepts/overview/what-is-kubernetes – Acessado em 01/03/2021.

MQTT (2021). Mqtt - the standard for iot messaging. Technical report, https://mqtt.org – Acessado em 15/03/2021.

Raspberry Pi Foundation (2021). Sense hat emulator. Technical report, https://sense-emu.readthedocs.io/en/v1.1/ – Acessado em 01/04/2021.

Xu, L. D., He, W., and Li, S. (2014). Internet of things in industries: A survey.

Yang, C., Shen, W., and Wang, X. (2016). Applications of Internet of Things in manufacturing. In *Proceedings of the 2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2016*, pages 670–675. Institute of Electrical and Electronics Engineers Inc.