A Multi-Agent Approach to Monitor and Manage Container-Based Distributed Systems

V. Pfeifer, W. F. Passini, W. F. Dorante, I. R. Guilherme and F. J. Affonso

***Abstract*—The software architectures' evolution over the years has brought new challenges for the software development area. New architectural styles have emerged, such as microservice architecture and container-based distributed systems. In this sense, it can be said that service monitoring systems can be considered as an important element, since they can observe and verify the quality and progress of an application over a period of time. Automated solutions based on pre-defined intervals can result in unnecessary use of resources to identify any type of anomalies (e.g., failure or Quality of Service degradation). Considering this scenario, a multi-agent approach to monitor container-based distributed systems is proposed in this article. Such approach autonomously and dynamically decides the monitoring frequency of a container, optimizing the identification of anomalies. To evaluate the applicability of our approach, we conducted a case study on an application to manage a smart restaurant named App2SmartRest. The results reported in this article enable us to create a favorable scenario for our approach to contribute efficiently with different software development communities.**

***Index Terms*—container-based applications, monitoring, multi-agent, learning techniques, distributed systems.**

I. Introdução

Atualmente, nossa sociedade está cada vez mais dependente de sistemas de software que devem ser capazes de operar em situações adversas e funcionar em modo 24/7 (ou seja, 24 horas por dia, sete dias por semana). Nesse sentido, nota-se que a complexidade de tais sistemas e de seus ambientes computacionais tem aumentado ao longo dos anos. Em síntese, esses sistemas lidam com estruturas mais complexas que permitem interpretar o contexto em que estão inseridos e, ao mesmo tempo, com requisitos extras que os tornam mais versáteis, flexíveis, extensíveis, resilientes, confiáveis, robustos, recuperáveis, customizáveis e configuráveis [1], [2], [3], [4], [5].

Os sistemas baseados em SOA (do inglês, *Service-Oriented Architecture*) têm desempenhado um papel importante no desenvolvimento de aplicativos distribuídos pela Internet [6]. De acordo com Lewis & Fowler [7], MSA (do inglês, *MicroService Architecture*) é um termo arquitetural que pode ser tratado como uma evolução das aplicações baseadas em SOA. Esse tipo de aplicação se caracteriza essencialmente pela modularização do software em pequenos serviços vinculados ao contexto em que atuam. Esses serviços se comunicam por meio de mecanismos leves, onde cada serviço pode ser desenvolvido na tecnologia mais adequada para a necessidade de negócio que o mesmo soluciona. A adoção de contêineres se tornou a solução *de facto* para aplicações baseadas na MSA por facilitar o empacotamento e a implantação dos microsserviços. O crescimento da quantidade de microsserviços/-contêineres no ambiente de execução torna inviável o monitoramento pelo ser humano, fazendo-se necessário a utilização de abordagens de monitoramento automatizadas. Em resumo, o principal propósito dessas abordagens é coletar dados sobre os recursos de interesse de monitoramento para que decisões possam ser tomadas no menor intervalo de tempo possível, visando mitigar ou evitar problemas que possam comprometer a execução de uma aplicação. Abordagens de monitoramento podem ocorrer em diferentes domínios de sistemas de software e utilizar diversas tecnologias. Por exemplo, no trabalho de Dominguez *et al.* [8], uma plataforma multiagente chamada SQ-MAS (do inglês, *Service Quality Multi-Agent System*) foi desenvolvida para monitorar e avaliar a qualidade do serviço em redes de distribuição de energia elétrica. Fontes *et al.* [9] desenvolveram uma arquitetura multiagente para acompanhamento de alunos em ambientes virtuais de aprendizagem, visando identificar a participação efetiva dos mesmos na modalidade de ensino a distância. No domínio de aplicações distribuídas baseadas em microsserviços e contêineres, abordagens de monitoramento podem ser consideradas como um elemento essencial. Nesse tipo de ambiente/aplicação, uma falha em um serviço pode desencadear falhas em outros serviços com o qual este se comunica. Portanto, detectar tais falhas o mais rápido possível a um custo de recursos reduzido pode ser considerado o cenário ideal para tais aplicações. De acordo com Ye *et al.* [10], [11], uma solução ingênua para esta problemática seria monitorar todos os serviços do sistema de maneira constante, resultando em um custo elevado de recursos. Assim, quanto maior a quantidade de monitoramentos necessários para encontrar uma falha, maior é o custo de monitoramento. Uma anomalia no sistema pode causar a degradação de QoS (do inglês, *Quality of Service*) e, consequentemente, a redução no tempo de resposta ou indisponibilidade do serviço. Portanto, quando uma anomalia é detectada ou prevista, ações de correção podem ser tomadas para manter a qualidade do sistema.

Baseado no contexto exposto, este artigo propõe uma abordagem multiagente para o monitoramento e gerenciamento de sistemas distribuídos baseados em contêineres. Essa abordagem tem a capacidade de definir, de maneira autônoma e dinâmica, a frequência de monitoramento de um

Vinicius Pfeifer, CI&T - Rua. Dr. Ricardo Benetton Martins, 1.000 – Pólis de Tecnologia – Prédio Prisma – 13086-902, Campinas, São Paulo. Mestrando na Universidade Estadual Paulista (UNESP), Brasil – Departamento de Estatística, Matemática Aplicada e Computação, Rio Claro, São Paulo, Brasil, 13506-900. e-mail: vpfeifer20@gmail.com.

William Filisbino Passini, William Fernandes Dorante, Ivan Rizzo Guilherme, and Frank José Affonso, Universidade Estadual Paulista (UNESP), Brasil – Departamento de Estatística, Matemática Aplicada e Computação, Rio Claro, São Paulo, Brasil, 13506-900. e-mail: william.passini@unesp.br, williamfernandes815@gmail.com, ivan.guilherme@unesp.br, f.affonso@unesp.br.

microsserviço/contêiner em execução. Até onde sabemos, embora hajam alguns esforços que apresentem algumas iniciativas nessa direção, nenhuma solução fornece cobertura completa e coerente de modo que seja possível o monitoramento eficiente, identificando e recuperando os microsserviços/contêineres que apresentam algum tipo de anomalia (ou seja, falha ou degradação de QoS) [10], [11], [12], [13], [14]. Nesse sentido, pode-se dizer que a abordagem proposta neste artigo é uma solução factível para o monitoramento de sistemas distribuídos baseados em contêineres. Diante do exposto, as principais contribuições deste trabalho podem ser sintetizadas nos seguintes itens:

- Projeto de uma abordagem multiagente para apoiar a atividade de monitoramento em sistemas distribuídos baseado em contêineres;
- Desenvolvimento de uma estratégia inteligente para monitoramento desse tipo de sistema, que decide de maneira autônoma e dinâmica a frequência de monitoramento de contêineres. Como resultado, pode-se tornar mais eficiente a utilização de recursos em relação ao monitoramento baseado em intervalos pré-definidos; e
- Projeto baseado em agentes como uma solução modular, onde é possível adaptar algoritmos e funções que lidam com a eficiência do monitoramento. Dessa foram, espera-se reduzir os esforços dos interessados quando nossa abordagem é adaptada para atuar em outros domínios de sistemas de software que requerem a atividade de monitoramento.

O restante deste artigo está organizado da seguinte maneira: na Seção II são apresentados os principais conceitos utilizados neste trabalho; na Seção III são reportados os principais trabalhos relacionados; na Seção IV é apresentada a descrição da abordagem proposta neste artigo; na Seção V é reportada a avaliação de nossa abordagem e, por fim, na Seção VI são sintetizadas nossas conclusões e perspectivas para pesquisas futuras.

II. Conceitos e Definições

Nesta seção são apresentados os principais assuntos que contribuíram para o desenvolvimento de nossa abordagem.

Agentes Inteligentes. Segundo Russel e Norvig [15], um agente pode ser considerado algo que consegue perceber o ambiente por meio de sensores e agir por meio de atuadores. Por exemplo, um agente humano possui olhos como sensores e mãos como atuadores. Já um agente robótico possui câmeras como sensores e motores como atuadores. Um agente de software recebe sequências de caracteres, conteúdo de arquivos e/ou pacotes de rede como entradas sensoriais, e atua sobre o ambiente exibindo informações na tela, escrevendo em arquivos e enviando pacotes de rede. De maneira geral, um sistema baseado em agentes inteligentes é desenvolvido para atuar em um ambiente, que pode ser completamente ou parcialmente observável. Esses sistemas podem ser formados por um ou mais agentes (multiagentes), cujas características dos ambientes em que são aplicados podem variar entre determinísticos ou estocásticos, episódicos ou sequenciais, estáticos ou dinâmicos, discretos ou contínuos, e conhecidos ou desconhecidos. Por exemplo, um agente projetado para dirigir um carro atua em um ambiente parcialmente observável, multiagente, estocástico, sequencial, dinâmico e contínuo.

Contêiner. Um contêiner pode ser definido como um método de virtualização no nível de sistema operacional que permite executar uma aplicação e suas dependências em processos com recursos isolados [16]. De acordo com a Docker Inc. [17], "um contêiner é uma unidade padrão de software que empacota o código e todas as suas dependências para que o aplicativo seja executado de forma rápida e confiável de um ambiente de computação para outro". O tempo de inicialização e implantação de aplicações baseadas em contêineres é inferior se comparado com máquinas virtuais. Outros benefícios da utilização de contêineres são: (i) consistência de ambiente, uma vez que todas as dependências necessárias para execução da aplicação estão encapsuladas no contêiner; (ii) eficiência operacional, pois os recursos de infraestrutura podem ser reduzidos; e, (iii) aumento de produtividade e controle de versões da aplicação. Dentre as soluções disponíveis no mercado, o Docker é uma das ferramentas mais utilizadas para criação, execução e publicação de contêineres por ser uma solução de código aberto, focada em DevOps (do inglês, *Development e Operations*) e na integração contínua, entre outros motivos [17].

Sistemas Distribuídos e Microsserviços. Soldani *et al.* [18] comentaram que uma extensão natural de SOA pode ser a MSA, uma vez que enfatiza que o gerenciamento de serviços seja realizado de maneira independente e autocontido. Em síntese, cada microsserviço é construído em torno de uma capacidade de negócio e executado em seu próprio processo, comunicando-se com os outros microsserviços por meio de mecanismos leves como, por exemplo, o protocolo HTTP (do inglês, *HyperText Transfer Protocol*) [19].

De acordo com Lewis e Fowler [7], as características principais de uma MSA podem ser sintetizadas em: (i) componentes via serviços, (ii) organização em torno de capacidades de negócios, (iii) foco em produtos e não projetos, (iv) *endpoints* inteligentes e tubos mudos, (v) governança descentralizada, (vi) gerenciamento de dados descentralizado, (vii) automação de infraestrutura, (viii) projeto orientado a resiliência, e (ix) design evolucionário. Finalmente, Hamzehloui *et al.* [20] reporta que os microsserviços são resultados de lições aprendidas de várias áreas da engenharia de software (por exemplo, *Domain Driven Design*, SOA, DevOps, Implantação Contínua) para muitos dos problemas de entrega e implantação.

III. Trabalhos Relacionados

Um modelo para monitoramento de infraestrutura em um sistema distribuído baseado em contêineres foi desenvolvido por Ciuffoletti [12]. Este modelo foi baseado nas diretrizes da OCCI (do inglês, *Open Cloud Computing Interface*) para a definição de uma interface para monitoramento de serviços e microsserviços baseada em agentes e sondas. Em síntese, os dados são coletados do ambiente para cada componente arquitetônico que necessita de monitoramento. No contexto

de infraestruturas de computação, carga de trabalho de processadores, espaço livre para armazenamento, e número de pacotes encaminhados para um dispositivo de rede podem ser considerados como exemplos de componentes e fontes de dados a serem coletadas. Em seguida, tais dados são processados e transformados em métricas especializadas para outras aplicações como, por exemplo, *dashboards*. Os conceitos de monitoramento baseado em agentes e a segmentação de responsabilidades por meio de componentes com funcionalidades específicas foram utilizados no desenvolvimento de nossa abordagem. Além disso, nossa abordagem utiliza um algoritmo de aprendizagem para tornar o monitoramento mais eficiente. Outro detalhe a ser destacado é que as ações são executadas no ambiente de execução da aplicação monitorada, de acordo com os dados coletados no monitoramento.

Wan *et al.* [21] propuseram um *framework* que visa minimizar o custo de implantação em aplicações baseadas em contêineres. No contexto do estudo realizado foram consideradas as seguintes restrições: (i) ambiente heterogêneo das aplicações baseadas em microsserviços, (ii) disponibilidade de recursos, (iii) custo de implantação de contêineres, e (iv) gerenciamento de recursos e aplicações. O *framework* proposto analisa os requisitos de aplicativos baseados em microsserviços e recursos disponíveis em máquinas físicas (por exemplo, *data centers* na nuvem) para alocação/substituição nas aplicações em desenvolvimento/execução. Como resultado, são determinadas as posições dos contêineres e suas respectivas tarefas. Por fim, vale destacar que o algoritmo embutido nessa solução funciona de maneira distribuída e incremental, tornando-o escalonável para recursos físicos massivos em diversas aplicações no *framework*. Diante do exposto, conceitos sobre a eficiência do monitoramento, visando reduzir o esforço e custo computacional, foram utilizados no projeto de nossa abordagem.

Uma abordagem de monitoramento de serviço baseada em agentes foi desenvolvida por Ye *et al.* [10], [11]. Tal abordagem visa lidar com sistemas SBS (do inglês, *Service-Based Systems*) em um contexto descentralizado. Os autores comentam que essa abordagem pode evitar o ponto único de falha e equilibrar o cálculo sobre os agentes de monitoramento. Em nossa abordagem (veja Seção IV) utilizamos o mesmo algoritmo proposto pelos autores supracitados modificando a função de importância para utilização de métricas de QoS, diferentemente da função original, a qual se baseia na quantidade de usuários. Outro fator que diferencia nossa abordagem da proposta de Ye *et al.* diz respeito ao contexto de aplicação do algoritmo, uma vez que nossa abordagem explora contêineres e microsserviços, enquanto Ye *et al.* avaliam o algoritmo em sistemas com múltiplos inquilinos (*multi-tenant*).

Imdoukh et al. [13] propuseram uma abordagem proativa baseada em aprendizado de máquina para realizar o escalonamento automático de contêineres Docker em resposta às mudanças dinâmicas de carga de trabalho em tempo de execução. Para isso, essa abordagem utiliza o *loop* de controle MAPE-K (do inglês, *Monitor, Analyze, Plan, Execute over a shared Knowledge*) [22]. Em síntese, tal abordagem visa prever a carga de trabalho futura para determinar o número de contêineres necessários para lidar com as solicitações com antecedência, eliminando atrasos causados pela inicialização ou interrupção da execução de contêineres. Nós adaptamos esse conceito para nossa abordagem quando tornamos o monitoramento de contêineres mais eficiente, evitando o uso desnecessário de recursos e, até mesmo, a interrupção da aplicação. Outra diferença importante é que nossa abordagem não trata da escalabilidade da aplicação, mas da manutenção dos contêineres configurados em funcionamento.

Um esquema de monitoramento para aplicações baseadas em microsserviços foi proposto por Jiang et al. [14]. De acordo com os autores, a solução proposta visa monitorar de maneira efetiva o funcionamento de múltiplos microsserviços, descobrindo problemas de maneira precoce para que soluções sejam propostas. Para isso, essa solução possui um sistema de alarme inteligente que combina dados de monitoramento coletados através de recursos disponibilizados pelos *frameworks* SpringBoot e SpringCloud. Em síntese, a solução proposta utiliza informações coletadas através de funcionalidades de *logging* e de informações de sensores configurados dentro de cada microsserviço, além de métricas sobre seu desempenho. Para nossa abordagem, evidências de monitoramento inteligente foram utilizadas, sendo empregados os conceitos que colaboram para um monitoramento mais eficiente e de maneira não invasiva (ou seja, sem injeção de código dentro dos microsserviços).

Apesar das iniciativas apresentadas nesta seção, trabalhos dedicados ao monitoramento de contêineres, que precisam de mais atenção em tempo de execução, ainda são objetos de interesse na academia e na indústria. De acordo com o exposto nesta seção, a eficiência na utilização de recursos e tempo na descoberta de falhas e/ou degradação de QoS tem se mostrado como principal interesse neste tipo de aplicação. Nesse sentido, procuramos destacar para cada trabalho as principais contribuições e diferenças em relação a abordagem proposta neste artigo.

IV. Abordagem Proposta

Esta seção apresenta a abordagem multiagente para monitoramento e gerenciamento de aplicações distribuídas baseadas em contêineres. Em síntese, uma aplicação baseada em contêineres que implementa uma MSA pode gerar dezenas, centenas ou milhares de contêineres, dependendo do tamanho do sistema. Dessa forma, um monitor de contêineres pode ser considerado um elemento de primeira classe para tais aplicativos. Basicamente, um monitor tem como objetivo observar e verificar a qualidade e o andamento de um aplicativo durante um intervalo de tempo. De modo geral, as ferramentas de monitoramento são responsáveis por coletar e analisar informações e métricas para rastrear o progresso da aplicação e de seus objetivos [11], [17], [23], [24]. Por esses motivos, monitorar esses contêineres manualmente não é uma tarefa viável por diferentes razões. Por exemplo, o uso de monitoramento automatizado com base em intervalos predefinidos pode resultar em desperdício de recursos. Uma maneira de contornar as adversidades supracitadas e fornecer uma alternativa viável capaz de tornar mais eficiente a utilização de recursos é a

abordagem proposta neste artigo. Essa abordagem visa, de maneira autônoma e dinâmica, definir uma frequência de monitoramento de contêineres eficiente, reduzindo a utilização de recursos para descoberta de falhas. A Fig. 1 ilustra nossa abordagem e, em seguida, uma descrição de cada componente é apresentada.

Inicialmente, **Passo A**, os engenheiros de software devem compreender o problema com base na especificação dos requisitos para que possam avaliar a viabilidade de desenvolvimento. Nesse passo, os microsserviços são projetados conforme diretrizes mencionadas por Lewis e Fowler [7] (veja Seção II). Em seguida, os microsserviços são desenvolvidos e os arquivos descritores de contêineres em arquivos JSON (do inglês, *Java Script Object Notation*) são criados. As informações contidas nesses arquivos são utilizadas pelos agentes de nossa abordagem para monitorar, criar, remover, iniciar e parar os contêineres de uma aplicação. A Listagem 1 mostra um exemplo de um arquivo descritor de contêiner. Na linha 2, o engenheiro de software deve atribuir um identificador para a aplicação. Nas linhas 3 e 4, devem ser informados o nome do microsserviço e de sua respectiva imagem. Finalmente, as linhas 5 e 6 especificam a porta do *host* em que a aplicação será disponibilizada e a porta usada internamente pelo contêiner.

Listagem 1. Arquivo descritor de contêiner

```
1 {
2  "applicationID" : "App2SmartRest"
3  "name"          : "customerMicroservice",
4  "image"         : "springboot/java:springJpaApp",
5  "hostPort"      : "8000",
6  "containerPort" : "80"
7 }
```

Nos **Passos B e C**, os engenheiros de software trabalham no projeto e implantação de cada aplicação, com base em diretrizes de desenvolvimento citadas na Seção II e boas práticas de Engenharia de Software [23], [25], [26]. Para isso, os microsserviços que irão compor o sistema devem ser especificados no arquivo descritor do contêiner. Este arquivo deve ser armazenado no **Repositório de Descritores de Contêineres** identificado por um ID para cada microsserviço, o qual será disponibilizado aos agentes de nossa abordagem. A seguir, detalhes de cada agente de nossa abordagem são apresentados.

O **Agente de Serviço** é responsável por monitorar o **Repositório de Descritores de Contêineres** do sistema. Na inicialização, esse agente verifica o repositório e carrega as configurações dos arquivos descritores em sua base de crenças. O **Agente Contêiner** é então notificado para que os contêineres de cada microsserviço sejam criados. De modo análogo, quando o repositório é atualizado (ou seja, os arquivos descritores são adicionados ou removidos), o **Agente de Serviço** emite um sinal para o **Agente de Contêiner** para executar uma operação no ambiente. As operações que podem ser executadas pelo **Agente de Contêiner** no ambiente relacionam-se ao ato de iniciar, reiniciar, parar ou remover contêineres quando notificado para realizar tais ações.

O **Agente Monitor** é responsável por perceber o ambiente de contêineres e, por meio de tal percepção, atuar ou não nesse ambiente. A implementação desse agente foi realizada com base no algoritmo proposto por Ye *et al.* [10]. Em síntese, a tarefa de monitorar um serviço é modelada por um jogo de supervisão composto por dois jogadores, onde o supervisor é representado pelo **Agente Monitor** e cada microsserviço representa um trabalhador. Dessa forma, o **Agente Monitor** pode monitorar ou não um microsserviço, enquanto o mesmo pode apresentar ou não falha em tempo de execução. O Algoritmo 1 mostra a relação entre o supervisor e o trabalhador.

Algoritmo 1: Algoritmo que relaciona supervisor e trabalhador no jogo de supervisão.

```
Entrada: r, c, p, b
início
    recompensa = r;
    custoInspecionar = c;
    punicao = p;
    beneficioDescanso = b;
    repita
        leia decisao supervisor;
        leia estado trabalhador;
        se supervisor inspeciona e trabalhador
         trabalhando então
            supervisor = recompensa - custoInspecionar;
            trabalhador = -(punicao - beneficioDescanso);
        fim
        se supervisor inspeciona e trabalhador
         descansando então
            supervisor = - custoInspecionar;
            trabalhador = 0;
        fim
        se supervisor não inspeciona e trabalhador
         trabalhando então
            supervisor = - recompensa;
            trabalhador = beneficioDescanso;
        fim
        se supervisor inspeciona e trabalhador
         trabalhando então
            supervisor = 0;
            trabalhador = 0;
        fim
    até infinito;
fim
```

Para que o modelo funcione corretamente, algumas condições devem ser satisfeitas. A punição (`punicao`) deve sempre ser maior que o benefício de descansar (`beneficioDescanso`) para que o trabalhador não escolha sempre descansar. De maneira análoga, a recompensa (`recompensa`) deve sempre ser maior do que o custo da inspeção (`custoInspecionar`), caso contrário, o supervisor nunca inspecionaria o trabalhador. Além disso, o custo da inspeção (`custoInspecionar`) deve ser maior do que 0 (zero), caso contrário, o supervisor sempre inspecionará o trabalhador. Analogamente, o benefício de descansar (`beneficioDescanso`) também deve ser maior do que 0 (zero), caso contrário, o trabalhador sempre trabalhará.

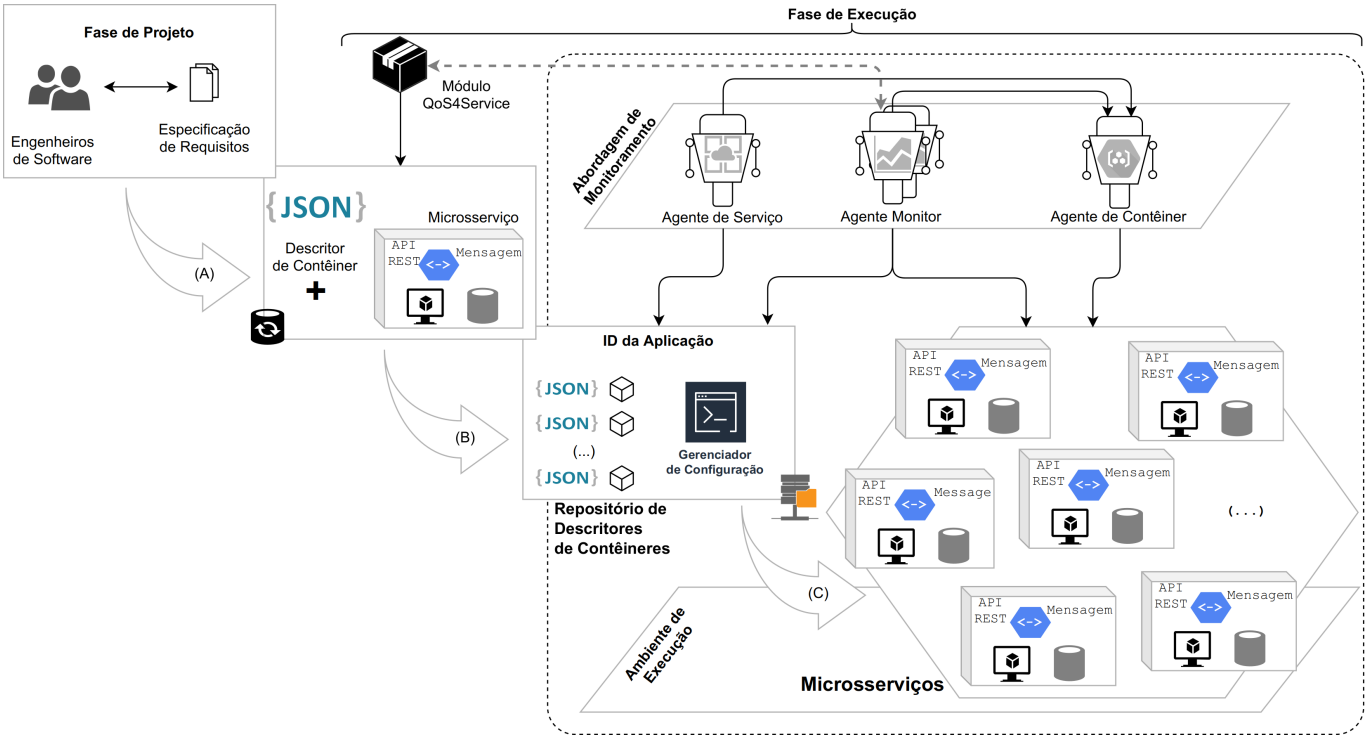

Fig. 1. Abordagem para gerenciamento de contêineres.

O estado desejado no problema de monitoramento é que o serviço esteja sempre trabalhando corretamente e o agente não precise monitorá-lo.

O modelo descrito é utilizado por Ye *et al.* [10] em um algoritmo de aprendizagem por reforço do tipo *Q-Learning*. Nesse tipo de algoritmo o agente não recebe conhecimento inicial, pois aprende por meio de tentativa e erro. Nossa abordagem implementa o algoritmo proposto na íntegra, exceto a função de importância, onde o algoritmo proposto utiliza uma fórmula baseada na quantidade de usuários por inquilino. Em contrapartida, nossa abordagem faz o uso de um módulo chamado QoS4Service para avaliação dos microsserviços, desenvolvido em trabalhos anteriores de nosso grupo de pesquisa [27], [28], [29]. Em síntese, esse módulo permite obter métricas de QoS para serviços do tipo SOAP (do inglês, Simple Object Access Protocol) e REST (do inglês, *Representational State Transfer*) [6], conforme interesses de uma aplicação. O QoS4Service fornece as métricas do serviço utilizadas na Equação 1, que representa a função de importância utilizada por nossa abordagem no algoritmo proposto por Ye *et al.* [10]. A pontuação (I_i) é obtida pela soma dos produtos entre o valor de determinado atributo de QoS "β" e seu respectivo peso "α", dentro do intervalo $[1, m]$, sendo m o total de atributos. Em relação ao peso α, vale ressaltar que os mesmos são definidos conforme interesses de cada aplicação [28].

$$I_{\mathrm{i}} = \sum_{i=1}^{m} \beta_i \times \alpha_i \tag{1}$$

O módulo QoS4Service também possui uma função de normalização (veja a Equação 2), a qual deve ser executada antes de se determinar uma pontuação final, garantindo que os resultados finais sejam coerentes. Nessa equação, β_i representa o valor que será calculado para cada um dos atributos, val_i representa o valor original do atributo avaliado e N representa o total de serviços necessários para a normalização dos dados. Essa função visa tratar situações em que determinados atributos sejam considerados melhores a partir da análise de um limite superior e outros por um limite inferior. Os atributos disponibilidade, latência e custo são considerados para ilustrar um cenário que um serviço possa ter valores elevados para disponibilidade e, ao mesmo tempo, valores o mais próximo possível de zero para latência e custo.

$$\beta_i = \begin{cases} \frac{val_i}{N}, \text{se limite inferior} \\ 1 - \frac{val_i}{N}, \text{se limite superior} \end{cases} \tag{2}$$

Por fim, vale destacar que o resultado da Equação 1 é levado em consideração pelo algoritmo para atualizar a probabilidade do **Agente Monitor**, que define se um microsserviço é ou não monitorado. Quando o *Agente Monitor* opta por monitorar um contêiner e uma falha é detectada, um sinal é emitido ao **Agente de Contêiner** para que o contêiner que contém aquele microsserviço seja reinicializado. O objetivo desse algoritmo em nossa abordagem é fazer com que o agente encontre uma probabilidade eficiente para o monitoramento de cada contêiner, de tal maneira que encontre as falhas em tempo hábil. Diferentemente do monitoramento baseado em intervalos, aquele que utiliza aprendizagem permite identificar o quão problemático um contêiner é para uma aplicação e o quanto é necessário monitorá-lo. Dessa forma, ao invés de realizar uma varredura completa em todos os microsserviços/contêineres de uma aplicação, uma análise mais eficiente por aqueles mais suscetíveis a falhas e/ou degradação de QoS é realizada.

V. Avaliação da Abordagem

Para avaliar a aplicabilidade, pontos fortes e fracos de nossa abordagem, esta seção apresenta o estudo de caso que conduzimos. Como objeto de nossa análise empírica, selecionamos um aplicativo chamado App2SmartRest, o qual visa apoiar a gestão de um restaurante inteligente. Esse aplicativo é resultado de experiências anteriores de nosso grupo de pesquisa [28], [29]. A seguir, apresentamos uma breve descrição de nossa aplicação e das estratégias empíricas adotadas para a realização deste estudo de caso.

Requisitos da aplicação. Em primeiro lugar, o App2SmartRest foi migrado para o estilo MSA baseado nas diretrizes estabelecidas por Dehghani [30] e experiências da indústria [18]. Por razões de escopo e espaço, detalhes de tal processo de migração não serão apresentados neste artigo. Em síntese, o App2SmartRest foi projetado para usar uma abordagem automatizada para atender seus clientes, onde são utilizados dispositivos embutidos nas mesas e luzes de controle. Inicialmente, quando o consumo é inicializado, o aplicativo do restaurante solicita a autenticação do cliente para abrir um pedido e a luz de controle da mesa muda de verde para vermelho. Uma vez autenticado, o cliente pode iniciar seu consumo fazendo pedidos, que são encaminhados para a fila de atendimento do restaurante. Quando o cliente optar por fechar o bloco de pedidos, o aplicativo mostra as opções de pagamento disponíveis. Se a opção de cartão de crédito ou débito for selecionada, o App2SmartRest permitirá que um ou mais cartões sejam inseridos na máquina de cartão de crédito acoplada à mesa para que o pedido possa ser pago. Finalmente, o aplicativo solicita que os clientes informem que estão se retirando da mesa. Em seguida, a luz de controle muda para a cor laranja e um funcionário do restaurante fará a limpeza e preparará a mesa para o próximo cliente. Além do fluxo normal de funcionamento, esta aplicação permite atender outras funcionalidades, tais como: cancelar um pedido, alterar mesas, consultar consumos, consultar detalhes de um prato, entre outras. A Fig. 2 ilustra a arquitetura de nossa aplicação.

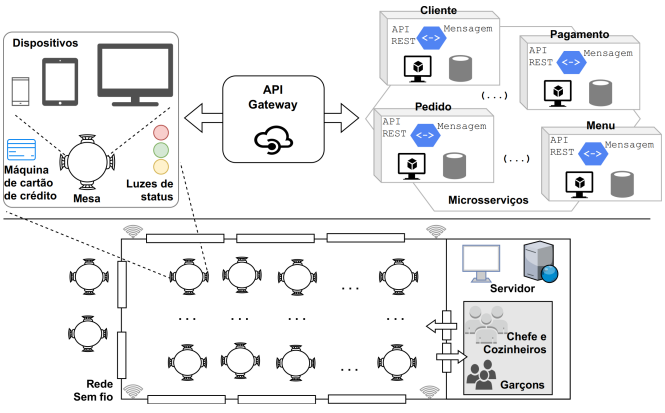

Fig. 2. Visão arquitetural do App2SmartRest.

Configuração e desenvolvimento. O App2SmartRest foi organizado em sete microsserviços, a saber: Cliente, Funcionário, Menu, Pedido, Pagamento, Serviço e Mesa. No que diz respeito ao desenvolvimento, esses microsserviços foram implementados com a combinação de diferentes tecnologias, a saber: Java [31] e MySQL [32] (por exemplo, microsserviços de Cliente e Pagamento); Java e MongoDB [33] (por exemplo, microsserviço de Produto); e .Net Core [34] e Elasticsearch [35] (por exemplo, microsserviços de Pedido e Menu). Em relação aos componentes de nossa abordagem, foram utilizadas as seguintes tecnologias: (i) Java; (ii) AgentSpeak, uma linguagem de programação orientada a agente baseada em lógica de programação e na arquitetura BDI (do inglês, *Belief-Desire-Intention*); (iii) JaCaMo [36] um *framework* para programação multi-agente que combina três tecnologias, a saber: (a) Jason [37], um interpretador para uma versão estendida de AgentSpeak; (b) CArtAgO [38], um *framework* de propósito geral para programação e execução de ambientes virtuais para sistemas multiagentes; e (c) Moise [39], um modelo organizacional para sistemas multiagente baseado nas noções de papéis, grupos e missões; (iv) Docker [17], uma plataforma de contêiner para microsserviços; e (v) Spotify Docker Client [40], um cliente Docker Java de código aberto desenvolvido pelo Spotify.

Para mostrar o comportamento de nossa abordagem, um **Agente de Teste** e um **Agente de Monitoramento em Intervalos** (AMI) foram desenvolvidos. O primeiro foi programado para interromper a execução de contêineres de maneira aleatória para simular um ambiente de execução com ocorrência de falhas. Durante a execução de nossa análise empírica, quatro cenários de falhas foram utilizados: 10, 100, 500 e 1000. Esses cenários foram selecionados visando mostrar o comportamento de nossa abordagem em relação ao número de falhas que podem ocorrer no ambiente de execução. Além disso, como nosso objetivo é analisar o comportamento do algoritmo de monitoramento e sua função de importância no monitoramento de microsserviços/contêineres, clientes para os microsserviços elaborados e a avaliação de QoS dos mesmos não foram considerados neste estudo de caso. Dessa forma, os engenheiros de software podem avaliar o comportamento do algoritmo adotado para monitoramento em um ambiente mais próximo da realidade, onde falhas ocorrem e como estas estão sendo identificadas. Nesse sentido, vale destacar que outros algoritmos podem ser acoplados à nossa abordagem (veja Seção IV) sem a necessidade de implementação adicional, pois nossa abordagem foi idealizada em princípios de reutilização (ou seja, uma interface comum está disponível para esses algoritmos) [41]. O AMI foi desenvolvido para monitorar os contêineres em intervalos definidos e constantes, ou seja, o agente sempre vai monitorar os contêineres a cada intervalo (`t`). O objetivo do AMI é servir como base de comparação com o **Agente Monitor** utilizando uma abordagem mais simples de monitoramento.

Execução e resultado. No que diz respeito ao comportamento da aplicação, dois cenários importantes foram selecionados, conforme ilustrado pelo diagrama de sequência na Fig. 3. O cenário de inicialização ocorre entre os Passos 1 e 10. Já o cenário de simulação ocorre entre os Passos 11 e 20, o qual utiliza os agentes supracitados (**Agente de Teste** e AMI). A seguir, uma descrição de cada cenário é apresentada.

A inicialização da aplicação ocorre no Passo 1 (método

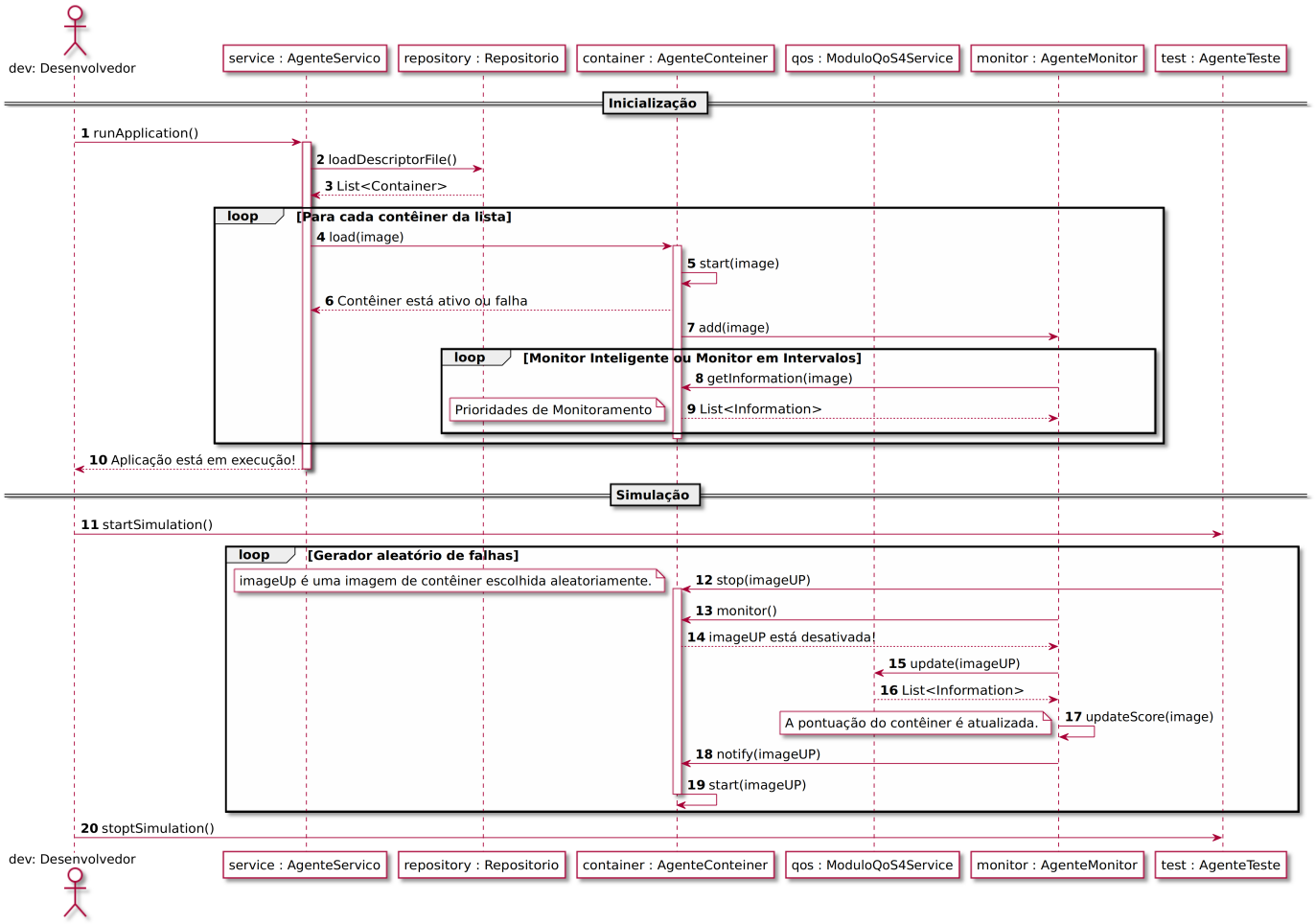

Fig. 3. Diagrama de sequência para simulação da abordagem de gerenciamento de contêineres.

`runApplication()`). Em seguida, os arquivos descritores são carregados (`loadDescriptorFile()`) e uma lista de contêineres é obtida (Passos 2 e 3). Entre os Passos 4 e 10 ocorre a ativação e monitoramento dos contêineres, os quais são carregados como imagem no Passo 5 (`start(image)`) e adicionados ao monitor (Passo 7) pelo método `add(image)`. O monitoramento de cada contêiner é realizado pelo **Agente Monitor** e ocorre nos Passos 8 e 9 (`getInformation(image)`).

O cenário de simulação mostra como os contêineres são interrompidos e inicializados para aparentar uma sequência de falhas durante a execução do aplicação (App2SmartRest) em um processo de seleção aleatória de microsserviços. Os contêineres são modificados para o estado inativo pelo **Agente de Teste** pelo método `stop(imageUP)` (Passo 12). O **Agente Monitor** identifica a mudança de estado pelo método `monitor()` (Passos 13 a 14) e notifica o **Agente de Contêiner** pelo método `notify(imageUP)` (Passo 18) para iniciar tal contêiner novamente pelo método `start(imageUP)` (Passo 19). Entre as etapas 15 a 17, a pontuação deste microsserviço é atualizada pelos métodos `update(imageUP)` e `updateScore(image)` para que receba mais atenção pela atividade de monitoramento.

As Figs. 4 e 5 mostram os resultados de uma análise comparativa entre o monitoramento baseado em Intervalo e em Inteligência Artificial (IA), que foi ilustrado na Fig. 3. A Fig. 4 apresenta a distribuição da quantidade de monitoramentos necessários para detecção de falha considerando nosso cenário mais amplo (ou seja, 1000 falhas). O número de monitoramentos médio para detecção de uma falha no monitoramento baseado em intervalos e em IA foram 9,9 e 4,84, respectivamente. Assim, nossa abordagem, identificada nos gráficos como IA, foi mais eficiente do que a baseada em intervalos, reduzindo a quantidade de monitoramento necessária para descoberta de uma falha em aproximadamente 50%. A quantidade de monitoramentos média para detecção de falha de cada abordagem, considerando os cenários supracitados, está ilustrada na Fig. 5. Como pode ser observado, quando comparada com a abordagem tradicional baseada em intervalos, nossa abordagem, identificada por IA nos gráficos, mostra-se mais vantajosa, característica que pode resultar na redução de tempo e de recursos computacionais necessários para identificar uma falha.

VI. Conclusão

Este artigo apresentou uma abordagem para apoiar o monitoramento e o gerenciamento de contêineres. Nossa abordagem usa agentes inteligentes e técnicas de aprendizagem, que

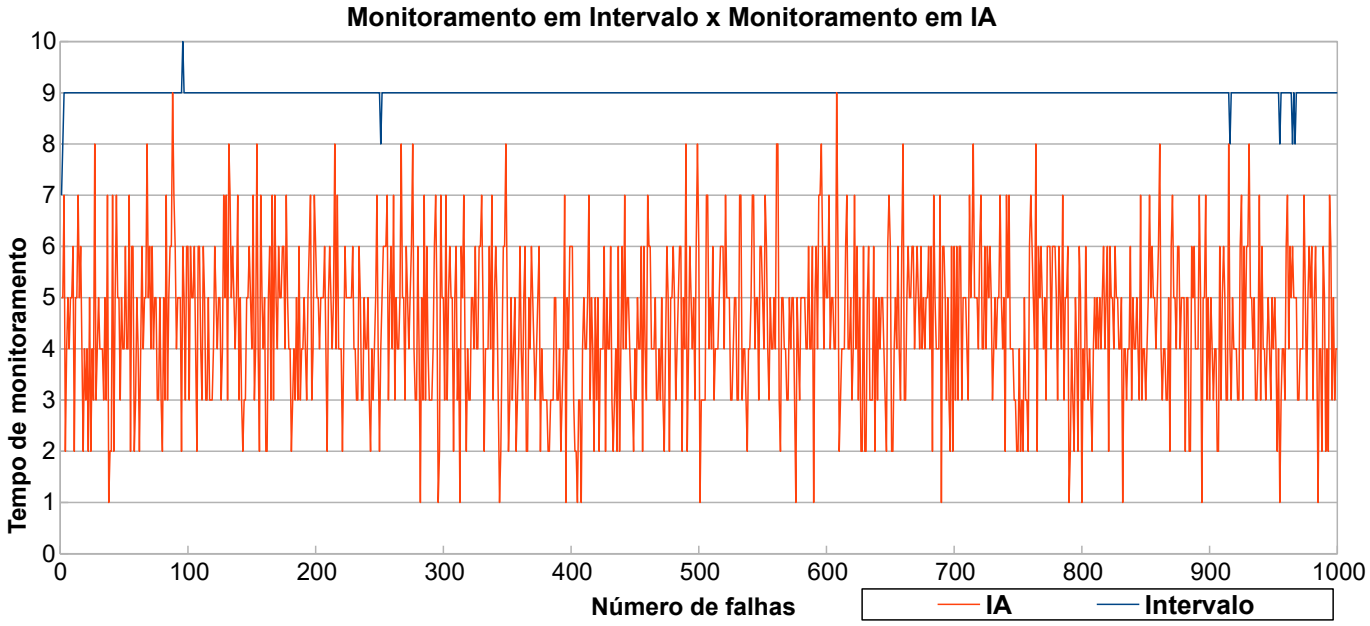

Fig. 4. Análise comparativa entre o monitoramento baseado em IA e em intervalos: Intervalo de 1000 falhas.

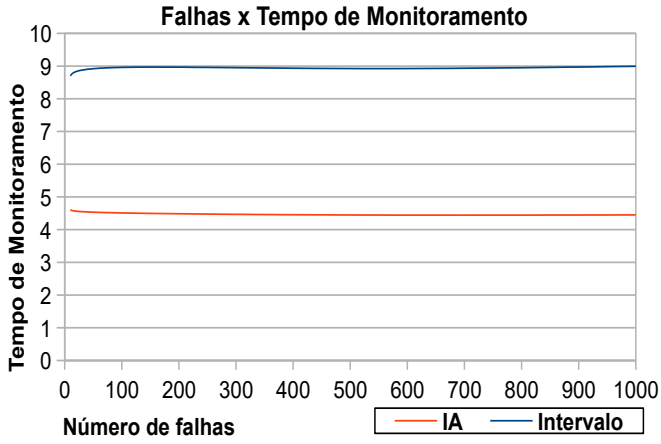

Fig. 5. Análise comparativa entre o monitoramento baseado em IA e em intervalos: Média entre 10 e 1000 falhas.

autonomamente e dinamicamente decidem uma frequência eficiente de monitoramento para um microsserviço/contêiner. Conforme relatado na Seção IV, o monitoramento de contêiner pode ser considerado um elemento importante para aplicativos distribuídos baseados em microsserviços/contêineres. O monitor visa observar e verificar a qualidade e o andamento de uma aplicação ao longo de seu ciclo de execução. Nesse sentido, nossa abordagem visa proporcionar melhorias no monitoramento de uma aplicação baseada em contêineres, direcionando os esforços para contêineres que apresentam maior índice de falhas ou degradação da qualidade. Com base no cenário apresentado, as principais contribuições deste artigo são para: (i) a área de aplicações distribuídas, fornecendo uma abordagem para apoiar o monitoramento de contêineres para diferentes domínios de software; e (ii) a área de computação orientada a serviços, uma vez que propomos uma abordagem que permite o monitoramento de contêineres por meio de um processo eficiente direcionado aos microsserviços mais problemáticos ou críticos.

Em relação aos trabalhos futuros, as seguintes atividades são pretendidas: (i) realização de mais estudos de caso ou prova de conceitos para avaliar nossa abordagem, incluindo diferentes domínios de software; (ii) realização de uma análise comparativa apoiada por métodos estatísticos entre os resultados de nossa abordagem com trabalhos similares existentes na literatura; e (iii) aplicação desta abordagem na indústria, uma vez que se pretende avaliar seu comportamento quando aplicada em ambiente real maior de desenvolvimento e execução. Portanto, com base no conteúdo apresentado neste artigo, pode-se idealizar um cenário positivo de pesquisa, permitindo que nossa abordagem se transforme em uma contribuição efetiva para as comunidades envolvidas.

AGRADECIMENTOS

Este trabalho teve apoio da Pró-Reitoria da UNESP de Pesquisa (PROPe/UNESP), da Coordenação do Aperfeiçoamento de Pessoal de Nível Superior (CAPES) N. do processo: PROEX-9259572/D, da Fundação de Amparo à Pesquisa do Estado de São Paulo (N. 2020/10288-5), e dos Projetos LINEAS e RADIAR (UNESP/Petrobras/FUNDUNESP Cooperação) N. 2017/00502-7 e N. 5850.0102453.16.9.

REFERENCES

[1] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[2] L. Bass, P. Clements, e R. Kazman, *Software Architecture in Practice*, 3rd ed. Boston, USA: Addison-Wesley Professional, 2012.

[3] M. Vargas Santiago, S. E. Pomares Hernandez, L. A. Morales Rosales, e H. Hadj Kacem, "Fault tolerance approach based on checkpointing towards dependable business processes," *IEEE Latin America Transactions*, vol. 14, no. 3, pp. 1408–1415, 2016.

[4] M. Vargas-Santiago, L. Morales-Rosales, S. Pomares-Hernández, e K. Drira, "Autonomic web services enhanced by asynchronous checkpointing," *IEEE Access*, vol. 6, pp. 5538–5547, 2018.
[5] M. Vargas-Santiago, L. Morales-Rosales, R. Monroy, S. Pomares-Hernandez, e K. Drira, "Autonomic web services based on different adaptive quasi-asynchronous checkpointing techniques," *Applied Sciences*, vol. 10, no. 7, 2020.
[6] T. Erl, *Service-Oriented Architecture: Analysis and Design for Services and Microservices*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2016.
[7] J. Lewis and M. Fowler, "Microservices, a definition of this new architectural term," [*On-line*], 2021, disponível em: https://martinfowler.com/articles/microservices.html. Acessado em: December 12, 2021.
[8] J. S. Dominguez, A. Cerqueira Junior, D. S. Dominguez, D. Frias, e S. M. Iglesias, "Using a multi-agent system for monitoring indicators of quality of service in power distribution networks," *IEEE Latin America Transactions*, vol. 13, no. 4, pp. 1048–1054, 2015.
[9] L. M. O. F. Fontes, R. A. M. Valentim, F. M. Neto, e R. C. Souza, "A multi-agent architecture for monitoring tutoring activities in vles," *IEEE Latin America Transactions*, vol. 14, no. 10, pp. 4327–4333, 2016.
[10] D. Ye, Q. He, Y. Wang, e Y. Yang, "Detection of transmissible service failure in distributed service-based systems," *Journal of Parallel and Distributed Computing*, vol. 119, pp. 36–49, 2018.
[11] ——, "An agent-based service adaptation approach in distributed multi-tenant service-based systems," *Journal of Parallel and Distributed Computing*, vol. 122, pp. 11–25, 2018.
[12] A. Ciuffoletti, "Automated deployment of a microservice-based monitoring infrastructure," *Procedia Computer Science*, vol. 68, pp. 163–172, 2015.
[13] M. Imdoukh, I. Ahmad, e M. G. Alfailakawi, "Machine learning-based auto-scaling for containerized applications," *Neural Computing and Applications*, vol. 32, no. 13, pp. 9745–9760, Jul 2020. [Online]. Available: https://doi.org/10.1007/s00521-019-04507-z
[14] Y. Jiang, N. Zhang, e Z. Ren, "Research on intelligent monitoring scheme for microservice application systems," in *2020 International Conference on Intelligent Transportation, Big Data Smart City (ICITBS)*, 2020, pp. 791–794.
[15] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009.
[16] I. Amazon Web Services, "Aws official website," [*On-line*], 2021, disponível em: https://aws.amazon.com/pt/containers/. Acessado em: December 12, 2021.
[17] D. Inc., "Docker official website," [*On-line*], 2021, disponível em: https://www.docker.com. Acessado em: December 12, 2021.
[18] J. Soldani, D. A. Tamburri, e W.-J. V. D. Heuvel, "The pains and gains of microservices: A systematic grey literature review," *Journal of Systems and Software*, vol. 146, pp. 215–232, 2018.
[19] C. Pahl, A. Brogi, J. Soldani, e P. Jamshidi, "Cloud container technologies: a state-of-the-art review," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2018.
[20] M. S. Hamzehloui, S. Sahibuddin, e K. Salah, "A systematic mapping study on microservices," in *Recent Trends in Data Science and Soft Computing*, F. Saeed, N. Gazem, F. Mohammed, e A. Busalim, Eds. Cham: Springer International Publishing, 2019, pp. 1079–1090.
[21] X. Wan, X. Guan, T. Wang, G. Bai, e B.-Y. Choi, "Application deployment using microservice and docker containers: Framework and optimization," *Journal of Network and Computer Applications*, vol. 119, pp. 97–109, 2018.
[22] IBM, "An architectural blueprint for autonomic computing," [*On-line*], 2005, disponível em: https://goo.gl/wawGvi, Acessado em: December 12, 2021.
[23] C. Richardson, *Microservices Patterns With examples in Java*, 1st ed. Manning Publications, October 2018.
[24] O. Zimmermann, "Microservices tenets," *Computer Science - Research and Development*, vol. 32, no. 3, pp. 301–310, Jul 2017.
[25] R. S. Pressman, *Engenharia de Software - Uma Abordagem Profissional*. AMGH Editora, 2016, 8a Edição (15 de janeiro de 2016).
[26] I. Sommerville, *Engenharia de software*. Pearson, 2019, 10a Edição (22 de abril de 2019).
[27] W. F. Passini and F. J. Affonso, "A Framework to Support the Development of Self-adaptive Service-oriented Mobile Applications," in *The 30th International Conference on Software Engineering and Knowledge Engineering*, 2018, pp. 286–291.
[28] ——, "Developing self-adaptive service-oriented mobile applications: A framework based on dynamic deployment," *International Journal of Software Engineering and Knowledge Engineering*, vol. 28, no. 11n12, pp. 1537–1558, 2018.
[29] F. J. Affonso, W. F. Passini, e E. Y. Nakagawa, "A reference architecture to support the development of mobile applications based on self-adaptive services," *Pervasive and Mobile Computing*, vol. 53, pp. 33–48, 2019.
[30] Z. Dehghani, "How to break a monolith into microservices," [*On-line*], 2018, disponível em: https://martinfowler.com/articles/break-monolith-into-microservices.html. Acessado em: December 12, 2021.
[31] Oracle, "Oracle java," [*On-line*], 2021, disponível em: https://www.oracle.com/br/java. Acessado em: December 12, 2021.
[32] Oracle Corporation, "Site do banco de dados mysql - Área do desenvolvedor," [*On-line*], 2021, disponível em: https://dev.mysql.com/. Acessado em: December 12, 2021.
[33] I. MongoDB, "Site do banco de dados mongodb," [*On-line*], 2021, disponível em: https://www.mongodb.com. Acessado em: December 12, 2021.
[34] Microsoft, ".net. free. cross-platform. open source." [*On-line*], 2021, disponível em: https://dotnet.microsoft.com. Acessado em: December 12, 2021.
[35] Elasticsearch, "Site oficial do elasticsearch," [*On-line*], 2021, disponível em: https://www.elastic.co. Acessado em: December 12, 2021.
[36] JaCaMo, "Site oficial do projeto jacamo," [*On-line*], 2021, disponível em: http://jacamo.sourceforge.net. Acessado em: December 12, 2021.
[37] Jason, "Site oficial do jason," [*On-line*], 2021, disponível em: http://jason.sourceforge.net/wp/. Acessado em: December 12, 2021.
[38] Cartago, "Site oficial do cartago," [*On-line*], 2021, disponível em: http://cartago.sourceforge.net/. Acessado em: December 12, 2021.
[39] Moise, "Site oficial do moise," [*On-line*], 2021, disponível em: http://moise.sourceforge.net/. Acessado em: December 12, 2021.
[40] Docker Client, "Github do docker client," [On-line], 2021, disponível em: https://github.com/spotify/docker-client. Acessado em: December 12, 2021.
[41] E. Gamma, R. Helm, R. Johnson, e J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

Vinicius Pfeifer é formado em Ciência da Computação pela Universidade Estadual Paulista (Unesp). É mestrando do Programa de Pós-Graduação em Ciência da Computação pela Universidade Estadual Paulista (PPGCC/Unesp). É arquiteto de software na CI&T desde 2016. Tem experiência na área de Ciência da Computação com ênfase em Engenharia de Software nas seguintes áreas: Arquitetura de Software, Sistemas Baseados em Serviços, Arquitetura de Microsserviços e Desenvolvimento Web.

William Filisbino Passini é formado em Ciência da Computação pela Universidade Estadual Paulista (Unesp). Recebeu o título de mestre em Ciência da Computação pela Universidade Estadual Paulista (Unesp) em 2020. Atua como desenvolvedor de software na empresa Amdocs desde 2021. Tem experiência na área de Ciência da Computação com ênfase em Engenharia de Software nas seguintes áreas: *Frameworks*, Sistemas Baseados em Serviços, Microsserviços, Sistemas de Software Autoadaptativos, Computação Móvel, Web Semântica e Desenvolvimento Web.

William Fernandes Dorante é graduando em Ciência da Computação pela Universidade Estadual Paulista (Unesp). Tem experiência na área de Ciência da Computação com ênfase em Engenharia de Software nas seguintes áreas: Sistemas Baseados em Serviços e Desenvolvimento Web.

Ivan Rizzo Guilherme é formado em Ciência da Computação pela Universidade Federal de São Carlos (1985). Recebeu seu título de doutor em 1996 pela Universidade Estadual de Campinas (Unicamp). Livre docente em Inteligência Artificial Aplicada a Indústria do Petróleo pela Universidade Estadual Paulista. Realizou estágio de pós-doutorado em 2012 na Universidade de Lisboa, Portugal. Atualmente é Professor Adjunto na Universidade Estadual Paulista (Unesp). Docente do Programa de Mestrado em Ciência da Computação da UNESP. Coordena o Laboratório de Inteligência Artificial da UNESP - Rio Claro e o Laboratório de Inteligência Artificial Aplicada a Petróleo (LIAAP), onde desenvolve projetos de pesquisa na área de Petróleo.

Frank José Affonso é formado em Ciência da Computação. Recebeu seu título de doutor em 2009 pela Universidade de São Paulo (USP). Realizou pós-doutorado em 2013–2014 na USP. Atualmente é professor assistente doutor da Universidade Estadual Paulista (Unesp), Rio Claro/SP, Brasil. Tem experiência na área de Ciência da Computação com ênfase em Engenharia de Software nas seguintes áreas: Arquitetura de Referência, Sistemas Baseados em Serviços, Computação Móvel e Sistemas de Software Autoadaptativos. Orienta alunos de mestrado no PPGCC/Unesp. Tem participado como membro de comitês de programa em muitas conferências e revisor de diversos periódicos. É representante institucional da SBC (Sociedade Brasileira de Computação) em Rio Claro.