Pré-impressão
EasyChair

№ 468

Detecção e Diagnóstico de Anomalias para Microserviços baseados em contêineres com monitoramento de desempenho

Qingfeng Du, Tiandi Xie e Yu He

30 de agosto de 2018

Detecção e Diagnóstico de Anomalias para Microserviços baseados em Contêineres com Monitoramento de Desempenho

Qingfeng Du, Tiandi Xie, e Yu He

Escola de Engenharia de Software, Universidade de Tongji, Xangai, China
{du cloud, xietiandi, KAINLF3@TONGJI.edu.cn

Abstrato. Com tecnologias emergentes de contêineres, tais como Docker, aplicações baseadas em mi-cruzamento de serviços podem ser desenvolvidas e implantadas em ambiente de nuvens muito agiler. A confiabilidade desses microserviços é uma das maiores preocupações dos fornecedores de aplicações. Os comportamentos anômalos que podem levar a falhas inesperadas podem ser detectados com técnicas de desnudamento de técnicas anômalas. Neste artigo, um sistema de detecção de anomalias (ADS) é projetado para detectar e diagnosticar as anomalias em microserviços através de mono-torneamento e análise dos dados de desempenho em tempo real dos mesmos. O ADS proposto consiste em um módulo de monitoramento que coleta os dados de desempenho de recipientes, um módulo de processamento de dados baseado no mod- els de aprendizagem de máquinas e um módulo de injeção de falhas integrado para o treinamento desses modelos. O módulo de injeção de falhas também é usado para avaliar a detecção de anomalias e o desempenho do diagnóstico de nosso ADS. Clearwater, um Subsistema Multimídia IP virtual de código aberto, é usado para a validação de nosso ADS e os resultados experimentais mostram que o ADS proposto funciona bem.

Palavras-chave: Detecção de anomalias - Microserviços - Monitoramento de desempenho - Aprendizagem da máquina.

1 Introdução

Atualmente, cada vez mais aplicações Web são desenvolvidas em projetos de microserviços para melhor escalabilidade, flexibilidade e confiabilidade. Uma aplicação em microservice ap-proach consiste em uma coleção de serviços que é isolada, escalável e resiliente a falhas. Cada serviço pode ser visto como uma aplicação própria e estes serviços expõem seus pontos finais para a comunicação com outros serviços. Com a adoção de uma arquitetura de microserviços, muitos benefícios podem ser obtidos. Por exemplo, o software pode ser lançado mais rapidamente, e as equipes podem ser menores e focar em seu próprio trabalho.

Para gerar recursos isolados suficientes para um tal número de serviços, as seguintes técnicas de virtualização são amplamente utilizadas. As máquinas virtuais (MVs) são formas tradicionais de se obter a virtualização. Cada VM criada tem seu próprio sistema operacional (SO). Container é outra tecnologia emergente para virtualização que está ganhando popularidade sobre as VMs devido a seu peso leve, alto desempenho e maior escalabilidade[1]. E os contêineres criados compartilham o sistema operacional (OS).

O desenvolvimento de tecnologias de virtualização, especialmente a tecnologia de contêineres, contribuiu para a ampla adoção da arquitetura de microserviços nos últimos anos. E os prestadores de serviços começam a colocar maiores exigências sobre a confiabilidade desses microserviços. Acordos de Nível de Serviço (SLAs) são geralmente feitos entre prestadores de serviços e usuários para especificar a qualidade dos serviços prestados. Eles podem incluir vários aspectos tais como requisitos de desempenho e propriedades de confiabilidade[2]. E conseqüências graves podem ser causadas por uma violação de tais SLAs.

A detecção de anomalias pode nos ajudar a identificar padrões incomuns que não se con- matizam aos padrões esperados e o diagnóstico de anomalias pode nos ajudar a localizar a causa raiz de uma anomalia. Como a detecção e o diagnóstico de anomalias exigem uma grande quantidade de dados históricos, os prestadores de serviços têm que instalar muitas ferramentas de monitoramento em sua infra-estrutura para coletar dados de desempenho de seus serviços em tempo real.

Atualmente, há dois desafios principais enfrentados por estes micro prestadores de serviços. Em primeiro lugar, para os microserviços baseados em contêineres, que métricas devem ser monitoradas. Em segundo lugar, mesmo que todas as métricas sejam coletadas, como avaliar se os "be- haviors" da aplicação são anômalos ou não.

Neste documento, um sistema de detecção de anomalias (ADS) é proposto e pode abordar estes dois principais desafios de forma eficiente. O ADS proposto dá um tipo proto-tipo para os prestadores de serviços para detectar e diagnosticar anomalias para microserviços baseados em contêineres com monitoramento de desempenho.

O papel está organizado da seguinte forma: A Seção II revisa os antecedentes técnicos e algumas técnicas de detecção de anomalias amplamente utilizadas. A Seção III apresenta primeiramente nosso ADS e seus três componentes principais. A Seção IV apresenta em detalhes a implementação do ADS proposto. A Seção V apresenta os resultados de validação da ADS proposta no estudo do caso Clearwater. A Seção VI conclui a contribuição e discute o trabalho futuro.

2 Antecedentes e Trabalhos Relacionados

2.1 Backgroud

A arquitetura Microservice é um padrão de projeto de aplicação de nuvem que desloca a complexidade da aplicação monolítica tradicional para a infra-estrutura[3]. Em comparação com um sistema monolítico, o arhitech- ture baseado em microserviços cria um sistema a partir de uma coleção de pequenos serviços, cada um deles iso- laticado, escalável e resiliente a falhas. Os serviços comunicam-se através de uma rede usando interfaces de programação de aplicações (API) de linguagem agnóstica.

Os contêineres são virtualizações leves em nível de SO que nos permitem executar uma aplicação e suas dependências em um processo isolado de recursos. Cada componente funciona em um ambiente isolado e não compartilha memória, CPU ou o disco do sistema operacional (SO) do host[4]. Com cada vez mais aplicações e serviços implantados em ambientes hospedados em nuvem, a arquitetura de microserviços depende muito do uso da tecnologia de contêineres.

A detecção de anomalias é a identificação de itens, eventos ou observações que não estão em conformidade com um padrão esperado ou outros itens em um conjunto de dados[5]. Em um conjunto de dados normal

A correlação entre a carga de trabalho e o desempenho da aplicação deve ser estável e flutua significativamente quando as falhas são acionadas[6].

2.2 Trabalho Relacionado

Com a ampla adoção da arquitetura de microserviços e tecnologias de contêineres, o monitoramento e a avaliação do desempenho se tornam um tema quente para os pesquisadores de contêineres. Em [7], os autores avaliaram o desempenho dos microserviços baseados em contêineres em dois modelos diferentes com os dados de desempenho da CPU e da rede. Em [8][9], os autores forneceram uma comparação de desempenho entre um ambiente Linux nativo, containers Docker e KVM(máquina virtual baseada em kernel). Eles tiraram uma conclusão de que o uso do docker poderia alcançar uma melhoria por forma de acordo com as métricas de desempenho coletadas por suas ferramentas de benchmarking.

Em [2], os autores apresentaram sua abordagem de detecção de anomalias para os vícios dos serviços de nuvem. Eles implantaram uma aplicação de nuvem que consistia de vários serviços em várias VMs e cada VM executava um serviço específico. Os dados de desempenho de cada VM foram coletados e então, processados para detectar possíveis anomalias com base em técnicas de aprendizagem de máquinas. Em [6], os autores propuseram uma estrutura de diagnóstico automático de falhas chamada FD4C. A estrutura foi projetada para aplicações de nuvem e, na seção de última geração, os autores apresentaram quatro períodos típicos em sua estrutura FD4C, incluindo monitoramento do sistema, visualização de status, detecção de falhas e localização de falhas. Em [10][11][12], os autores prestaram atenção ao desempenho do sistema. Para detectar anomalias, eles construíram modelos com métricas históricas de desempenho e os compararam com modelos monitorados on-line. No entanto, estes métodos requerem conhecimento de domínio (por exemplo, a estrutura inter nal do sistema). Embora estes trabalhos enfoquem apenas o monitoramento em nível de VM e a detecção de falhas, eles nos dão muito que pensar e os métodos podem ser usados em microserviços baseados em contêineres de forma semelhante.

Este documento tem como objetivo criar um ADS que possa detectar e diagnosticar
anomalias para microserviços baseados em contêineres com monitoramento de desempenho. O ADS proposto consiste em três módulos: um módulo de monitoramento que coleta os dados de desempenho de containers, um módulo de processamento de dados que detecta e diagnostica anomalias e um módulo de injeção de falhas que simula falhas de serviço e reúne conjuntos de dados de desempenho que representam condições normais e anormais.

3 Sistema de Detecção de Anomalias

Esta seção apresenta uma visão geral de nosso sistema de detecção de anomalias. Há três módulos em nosso ADS. Primeiro, o módulo de monitoramento coleta os dados de monitoramento do desempenho do sistema alvo. Em seguida, o módulo de processamento de dados analisará os dados coletados e detectará as anomalias. O módulo de injeção de falhas simula falhas de serviço e reúne conjuntos de dados de monitoramento de desempenho que representam condições normais e anormais. Os conjuntos de dados são usados para treinar a aprendizagem da máquina

modelos, bem como para validar o desempenho de detecção de anomalias do ADS proposto.

Para a validação de nosso ADS, um sistema alvo composto de vários microserviços baseados em contêineres é implantado em nosso cluster de contêineres. Os dados de desempenho do sistema alvo são coletados e processados para a detecção de possíveis anomalias.

Normalmente, um usuário só pode visitar as APIs expostas da aplicação superior e não pode acessar diretamente o serviço específico implantado no motor do estivador ou na VM. Assim, nosso ADS não recebe nenhum conhecimento a priori sobre as características relevantes que podem causar comportamentos anômalos. O ADS proposto tem que aprender com os dados de monitoramento de desempenho com os próprios modelos de aprendizagem da máquina.

3.1 Agente de monitoramento

Uma aplicação baseada em contêineres pode ser implantada não somente em um único host, mas também em vários clusters de contêineres[13]. Cada cluster de containers consiste de vários nós (hospedeiros) e cada nó contém vários containers. Para aplicações implantadas em tais ambientes baseados em contêineres, os dados de monitoramento de desempenho devem ser coletados de várias camadas de uma aplicação (por exemplo, camada de nó, camada de contêiner e camada de aplicação). Nosso trabalho está focado principalmente no monitoramento de contêineres e no monitoramento de microserviços.

Monitoramento de contêineres Diferentes serviços podem ser adicionados em um único contêiner, mas na prática, é melhor ter muitos contêineres pequenos do que um grande. Se cada contêiner tem um foco restrito, é muito mais fácil manter seus microserviços e diagnosticar problemas. Neste artigo, o recipiente é definido como um grupo de um ou mais recipientes que constituem um microserviço completo, é a mesma definição de cápsula em Kubernetes. Ao processar os dados de desempenho de um contêiner, podemos dizer se o contêiner funciona bem.

Microservice monitoring Neste papel, um recipiente contém apenas um microserviço específico e um microserviço pode ser implantado em vários recipientes ao mesmo tempo. Ao coletar os dados de desempenho de todos os recipientes relacionados, podemos obter os dados de desempenho total de um micro-serviço específico. E também podemos saber se um microserviço é anômalo ao processar esses dados de desempenho do serviço.

3.2 Processamento de dados

Tarefas de processamento de dados O processamento de dados nos ajuda a detectar e diagnosticar anomalias - mentiras. Carla et al. definiram uma anomalia como a parte do estado do sistema que pode levar a um SLAV[2]. Usamos a mesma definição de anomalia que a indicada no trabalho de Carla. Uma anomalia pode ser uma CPU hog, vazamento de memória ou perda de embalagem de um recipiente que executa um microserviço, pois pode levar a um SLAV. Em nosso trabalho, há duas tarefas principais: classificar se um microserviço está passando por alguma anomalia específica e localizar o recipiente anômalo quando uma anomalia ocorre.

Modelos de processamento de dados As técnicas de detecção de anomalias são baseadas em algoritmos de aprendizagem ma-chine. Existem principalmente três tipos de aprendizagem de máquina al-goritmos: supervisionada, não supervisionada e semi-supervisionada. Todos estes algoritmos podem ser aplicados para classificar os comportamentos do sistema alvo com dados de monitoramento de desempenho.

Para detectar diferentes tipos de anomalias que podem levar a SLAVs, são utilizados algoritmos de aprendizagem super-visuais. Em nosso ADS, o algoritmo de aprendizagem supervisionada consiste em duas fases, mostradas na Fig. 1 e na Fig. 2.

A figura 1 mostra a fase de treinamento. Demonstra como os modelos de classificação são criados. Primeiramente, amostras de dados de desempenho rotulados que representam diferentes comportamentos de serviço são coletadas e armazenadas em um banco de dados. Estas amostras são chamadas de dados de treinamento. Em seguida, o módulo de processamento de dados treina os modelos de classificação com estes dados de treinamento. Para simular as solicitações reais dos usuários, um gerador de carga de trabalho é implantado. Para coletar mais dados de desempenho em diferentes tipos de erros, um módulo de injeção de falhas é implantado e ele injetará diferentes falhas em containers. Com mais amostras coletadas, o modelo será mais preciso.

A segunda fase é a fase de detecção. Uma vez que o modelo é treinado, alguns dados de desempenho em tempo real podem ser coletados e transferidos para o módulo de processamento de dados como entradas, e o módulo de processamento de dados pode detectar anomalias que ocorrem no sistema com o modelo treinado. Para a validação do módulo de processamento de dados, alguns erros serão injetados no sistema alvo, e então o módulo de processamento de dados usa os dados de desempenho em tempo real para detectar estes erros.

A anomalia de um serviço é freqüentemente causada pelos comportamentos anômalos de um ou mais recipientes pertencem a este serviço. Para descobrir se a anomalia é causada por algum recipiente específico, é utilizada a análise de séries temporais. Se vários contêineres executam um mesmo microserviço, eles devem fornecer serviços equivalentes aos usuários. A carga de trabalho e o desempenho de cada contêiner deve ser semelhante. Por este motivo, se uma anomalia for detectada em um microserviço, os dados de série temporal de todos os contêineres que executam este microserviço serão analisados. A similaridade entre os dados será medida e o contêiner anômalo será encontrado.

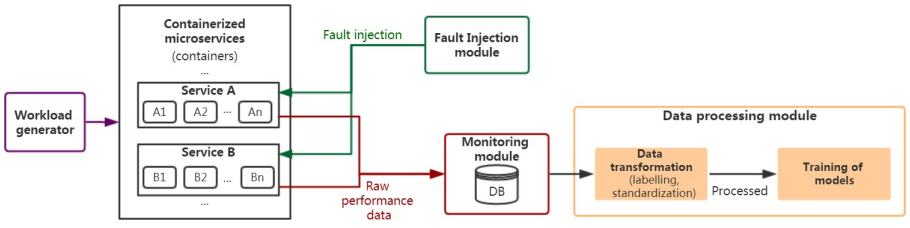

Fig. 1. Fase de treinamento, offline.

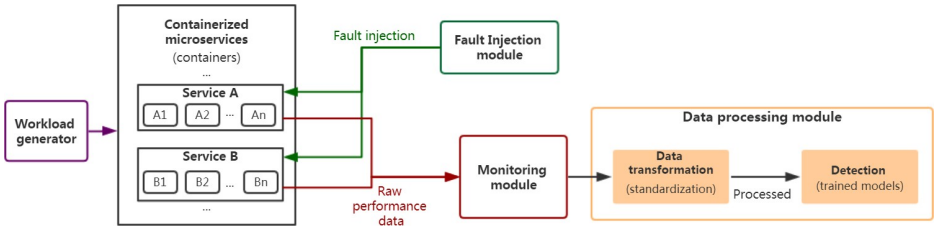

Fig. 2. Fase de detecção, online.

3.3 Injeção de falha

O módulo de injeção de falhas é integrado para coletar os dados de desempenho nas condições do sistema var- ious e treinar os modelos de aprendizagem da máquina. Para simular anomalias reais do sistema, nós escrevemos scripts para injetar diferentes tipos de falhas no sistema alvo. Quatro tipos de falhas são simulados com base nos recursos que impactam: alto consumo de CPU, vazamento de memória, perda de pacotes de rede e aumento da latência da rede.

Este módulo também é usado para avaliar a detecção de anomalias e o diagnóstico por formalidade de nosso ADS. Como mostrado na Fig. 2, após o treinamento dos modelos de classificação, o módulo de injeção de falhas injeta as mesmas falhas no sistema alvo, e os dados de desempenho em tempo real são processados pelo módulo de processamento de dados. Os resultados da detecção são usados para a validação.

4 Implementação

Esta seção apresenta a implementação dos três módulos do sistema de detecção de anomalias proposto.

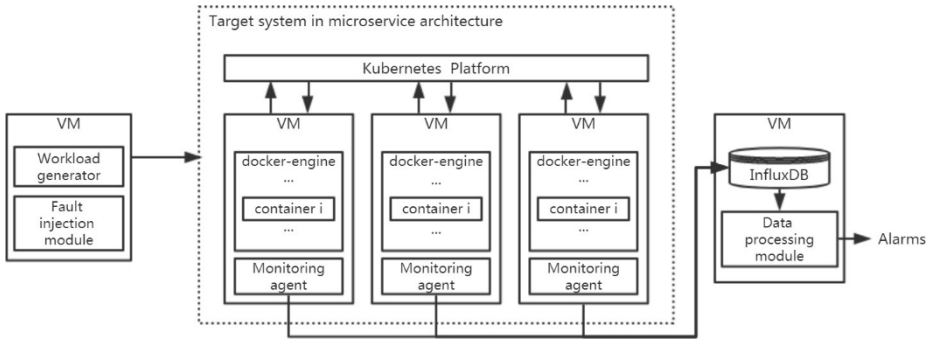

Fig. 3. Implementação do ADS.

Um protótipo do ADS proposto é implantado em uma plataforma virtualizada chamada Kubernetes. Como mostrado na Fig. 3, a plataforma é composta de várias VMs. As VMs são conectadas através de uma netowrk interna. Um sistema alvo em arquitetura de microserviço é implantado na plataforma para a validação e o sistema alvo consiste de vários containers rodando em diferentes VMs. O módulo de monitoramento instala um agente de monitoramento em cada VM para coletar dados de desempenho em tempo real e armazena os dados coletados em um banco de dados de série temporal chamado InfluxDB. O módulo de processamento de dados obtém os dados do banco de dados e executa tarefas de processamento com os dados. O módulo de injeção de falhas e o gerador de carga de trabalho funcionam executando scripts bash em outra VM.

4.1 Módulo de monitoramento

Como mostrado na Fig. 3, um agente de monitoramento é implantado em cada uma das VMs. Em um agente de monitoramento, várias ferramentas de monitoramento de código aberto são usadas para coletar e armazenar métricas de desempenho do sistema alvo, tais como cAdvisor e Heapster. O CAdvisor coleta os dados de utilização de recursos e monitoramento de desempenho de todos os contêineres enquanto o Heapster agrupa esses dados e armazena em um banco de dados de séries cronológicas chamado InfluxDB. As métricas da tabela 1 são coletadas para cada serviço e recipiente, incluindo métricas de CPU, métricas de memória e métricas de rede.

Tabela 1. Métricas de monitoramento

Nome métrico	Descrição
	cpu/usageCumulative CPU use em todos os núcleos.
cpu/pedidoCPU	(a quantidade garantida de recursos) em milicores. cpu/utilização
deCPU	em todos os núcleos em milicores.
cpu/limite	Limite rígido de CPU em milicores.
	memória/utilizaçãoTotal uso de memória.
memória/pedidoMemória	(a quantidade garantida de recursos) em bytes. memória/limite
	Limite rígido de memória em bytes.
	memória/conjunto de trabalhoTotal uso do conjunto de trabalho.
Conjunto de trabalho é a	memória
	sendo usado e não sendo facilmente derrubado pela amêndoa.
uso de	memória/cacheCache de memória.
	uso de memória/rssRSS
	memória. memória/page-
faultsNúmero de	páginas defeituosas.
memória/page-faults-rate	Número de páginas defeituosas
por segundo.	
rede/rx	Número cumulativo de bytes recebidos através da rede.
rede/rx-rate	Número de bytes recebidos através da rede por segundo.
rede/rx-errores	Número cumulativo de erros ao receber pela rede. rede/rx-
taxa de erros	Número de erros ao receber através da rede por segundo.
rede/tx	Número cumulativo de bytes enviados através da rede.
rede/taxa	Número de bytes enviados através da rede por segundo.
rede/tx-errores	Número cumulativo de erros durante o envio pela rede.
rede/tx-taxa de erros	Número de erros durante o envio pela rede.

4.2 Módulo de processamento de dados

O módulo de processamento de dados executa as duas tarefas para cada serviço, conforme discutido na Seção III. Os modelos de classificação são treinados com quatro algoritmos incluídos no scikit-learn da biblioteca. Os resultados são mostrados na Seção V.

- Máquinas vetoriais de suporte (configuradas com kernel=linear)
- Florestas Aleatórias (configuradas com profundidade máxima=5 e n estimadores=50)
- Naive Bayes
- Vizinhos mais próximos (configurados com k=5)

Após a fase de detecção, o serviço anômalo e o tipo de anomalia podem ser obtidos (por exemplo, CPU hog no Serviço A). A seguir, os recipientes anômalos devem ser diagnosticados. Se houver apenas um recipiente executando o serviço anômalo, ele pode ser diagnosticado como o recipiente anômalo diretamente. Entretanto, se vários recipientes estiverem executando o serviço anômalo, um algoritmo é necessário para diagnosticar o recipiente anômalo. O agrupamento de dados de séries temporais é uma boa solução e alguns algoritmos podem ser usados facilmente[14][15]. Entretanto, o agrupamento necessita de uma grande quantidade de dados, e as pessoas raramente implantam um número tão grande de contêineres. Neste caso, supomos que haja apenas um container anômalo ao mesmo tempo.

A distância entre duas seqüências temporais $\mathbf{x} = [x_1, x_2, ..., x_n]$ e $\mathbf{y} = [y_1, y_2, ..., y_n]$ pode ser computada muito facilmente através da distância euclidiana. Entretanto, a duração das duas seqüências temporais dadas deve ser a mesma. O algoritmo DTW é uma melhor escolha para medir a similaridade entre duas seqüências temporais. Ele encontra um alinhamento ótimo entre duas seqüências dadas, empena as seqüências com base no alinhamento, e então, calcula a distância entre elas. O algoritmo DTW tem sido usado com sucesso em muitos campos, como reconhecimento de fala e recuperação de informações.

Neste artigo, o algoritmo DTW é usado para medir a similaridade entre os dados de desempenho das séries temporais dos recipientes em questão. Uma vez detectada uma métrica anômala em um serviço, os dados das séries temporais de todos os contêineres que executam esse serviço serão analisados pelo algoritmo. E o contêiner mais anômalo que tiver a distância máxima dos outros será encontrado.

4.3 Módulo de injeção de falha

Um agente de injeção é instalado em cada recipiente de um serviço. Os agentes são executados e parados através de uma conexão SSH e eles simulam falhas de CPU, falhas de memória e falhas de rede por algumas implementações de software.

As falhas de CPU e memória são simuladas usando um software chamado Stress. A latência da rede e perda de pacotes são simuladas usando um software chamado Pumba.

Os procedimentos de injeção são projetados após a implementação dos agentes injetores. Para criar um conjunto de dados com vários tipos de anomalias em diferentes recipientes, um algoritmo é projetado e mostrado em 1. Após o procedimento de injeção ser concluído, os dados coletados são usados para criar conjuntos de dados de anomalias.

Algoritmo 1 Procedimento de injeção de falha.

Entrada: *lista de recipientes*, *lista de tipos de falhas*, *duração da injeção*, *tempo de pausa*, *carga de trabalho*

1: *GerarCarga de trabalho*(*carga de trabalho*)
2: **para** *lista de contêineres* em *contêineres* **fazer**
3: **para** *tipo_de falha* na *lista de tipos de falha* **fazem**
4: *injeção* = *Injeção*(*tipo de falha*, *duração da injeção*) 5: *injetar em recipiente*(*recipiente*, *injeção*) 6: *dormir*(*tempo de pausa*)
7: **fim para**
8: **fim para**

5 Estudo de caso

5.1 Descrição do ambiente

Kubernetes é um sistema de código aberto para automatizar a implantação, escalonamento e gerenciamento de aplicações em contêineres. O sistema alvo (Clearwater) funciona em uma plataforma kubernetes que consiste de três VMs (que são rain-u2, rain-u3 e rain-u4). Cada VM tem 4 CPUs, uma memória de 8 GB e um disco de 80 GB. As VMs são conectadas através de uma rede de 100 Mbps. Um agente de monitoramento é instalado em cada uma das VMs. As ferramentas de monitoramento instaladas incluem cAdvisor, Heapster, InfluxDB e Grafana.

Clearwater é uma implementação de código aberto do IMS (o Subsistema Multimídia IP) projetado desde o início para uma implantação massivamente escalável na Nuvem para fornecer serviços de voz, vídeo e mensagens a milhões de usuários[16]. Ele contém seis componentes principais: Bono, Sprout, Vellum, Homer, Dime e Ellis. Em nossa plataforma kubernetes, cada contêiner executa um serviço específico e pode ser facilmente escalonado. Neste documento, nosso trabalho está focado no Sprout, Cas- sandra e Homestead, que juntos constituem as Funções de Controle de Chamada/Sessão (CSCF), e realizamos experimentações para estes três serviços.

5.2 Experiências em águas claras

Antes de tudo, Clearwater é implantada em nossa plataforma kubernetes. Todos os serviços são executados em contêineres e o número de réplicas de componentes de homestead é fixado em três. Isso significa que haverá três contêineres rodando a mesma propriedade de serviço. Os dados de desempenho de um serviço são a soma de todos os contêineres rodando este serviço.

Em seguida, dois conjuntos de dados (conjunto de dados A e B) são coletados com a ajuda do módulo de injeção de falhas. Os procedimentos de injeção são mostrados na Tabela 2. Ao combinar os dois conjuntos de dados juntos, um terceiro conjunto de dados pode ser obtido como conjunto de dados
C. Após ser padronizado e rotulado, um conjunto de dados tem uma estrutura como mostrado na Tabela 3.

Como esses três serviços constituem a função CSCF juntos, haverá algumas relações entre seus dados de desempenho. E a questão se

podemos detectar as anomalias com os dados de desempenho de apenas um serviço. Para responder a esta pergunta, cada conjunto de dados é dividido em três conjuntos de dados menores de acordo com o serviço, e os algoritmos de classificação também são executados nestes conjuntos de dados para a validação. A estrutura do conjunto de dados dividido é mostrada na Tabela 4.

Tabela 2. Procedimentos de injeção de defeito

Procedimentos de injeção de experiência
lista de contêineres = {PRODUTO,cassandra,HOMESTEAD} tipo de falha = {CPU, memória,latência,PERDA de pacote} conjunto de dados A duração da injeção = 50min tempo de pausa = 10min carga de trabalho = carga de trabalhoA(5000 chamadas por segundo) lista de contêineres = {SPROUT,cassandra,HOMESTEAD} tipo de falha = {CPU, memória,latência,PERDA de pacotes} conjunto de dados Duração da binjeção = 30min tempo de pausa = 10min carga de trabalho = carga de trabalhoB(8000 chamadas por segundo)

Tabela 3. Estrutura do conjunto de dados

Tempo	Cassandra outro Homestead	Sprout	CPU com etiqueta Memmetrics métricas			
2018-05-08T09:21:00Z	512	70142771	...	...	...	nomal
2018-05-08T09:21:30Z	350	120153267	...	...	...	vazamento de cass mem
2018-05-08T09:22:00Z	322	70162617	...	...	....porco de cpu	broto

Tabela 4. Estrutura do conjunto de dados de serviço

TimeCassandra	CPU Cassandra	Mem outras métricas	etiqueta 2018-
05-08T09:21:00Z	51270142771	...	nomal
2018-05-08T09:21:30Z	350	120153267	... vazamento de cass mem
2018-05-08T09:22:00Z	322	70162617	...porco de cpu broto

Como injetamos quatro tipos diferentes de falhas em três serviços diferentes, haverá 12 rótulos diferentes. Também coletamos os dados em condições normais e em uma carga de trabalho pesada, portanto, há 14 etiquetas diferentes totalmente nestes conjuntos de dados.

5.3 Resultados da validação

Detecção de serviço anômalo Quatro algoritmos amplamente utilizados são comparados neste trabalho para treinamento dos modelos de classificação de nossos conjuntos de dados, que são Support Vector Machine(SVM), Nearest Neighbors(kNN), Naive Bayes(NB) e Random Forest(RF). O objetivo destes classificadores é descobrir o serviço anômalo com os dados de desempenho monitorados.

Há 757 registros no conjunto de dados A, 555 registros no conjunto de dados B e 1312 registros no conjunto de dados C. Para cada conjunto de dados, 80% dos registros são usados como conjunto de treinamento para treinar o modelo de classificação e os restantes 20% são usados como conjunto de teste para validar o modelo. Os resultados da validação são mostrados nas tabelas 5 e 6.

Com relação aos resultados de validação na Tabela 5, o desempenho de detecção do serviço anômalo é excelente para a maioria dos classificadores com valores de medida acima de 0,9. Para o conjunto de dados A, todos os quatro classificadores dão excelentes resultados de validação. Para o conjunto de dados B, três destes classificadores dão resultados maravilhosos, exceto SVM. Para o conjunto de dados C, o desempenho da Floresta Aleatória e dos Vizinhos mais Próximos ainda é excelente. Estes resultados mostram que o conjunto de dados criado por nosso ADS é significativo, e ambos os classificadores de Floresta Aleatória e Vizinhos Mais Próximos têm um excelente desempenho de detecção.

Para responder à pergunta se as anomalias podem ser detectadas a partir dos dados por formalidade de apenas um serviço relacionado, realizamos os mesmos experimentos nos três conjuntos de dados divididos do conjunto de dados C. Os resultados da classificação dos conjuntos de dados di-vistos (mostrados na Tabela 6) não são tão bons quanto os resultados usando o conjunto de dados completo. No entanto, o classificador de Vizinhos mais próximos ainda dá resultados satisfatórios em todos os três conjuntos de dados divididos. SVM parece ser o pior porque não tem um bom desempenho em conjuntos de dados com múltiplas classes. Consequentemente, o classificador de Vizinhos Mais Próximos é recomendado se você tiver que usar um conjunto de dados com apenas um serviço.

Tabela 5. Resultados da validação de três conjuntos de dados

Dataset	Medida	kNN	SVM	NB	RF
A	Precisão	0,93	0.95	0.95	0.95
	Relembrar	0.92	0.93	0.92	0.93
	F1-score	0,93	0.93	0.93	0.92
B	Precisão	0,98	0.75	0.98	0.99
	Relembrar	0.82	0.97	0.99	0.97
	F1-score	0,97	0.77	0.97	0.99
C	Precisão	0,96	0.82	0.83	0.93
	Relembrar	0.80	0.79	0.91	0.96
	F1-score	0,96	0.78	0.78	0.91

Diagnóstico de recipiente anômalo A anomalia de latência de rede do container homestead-1 é utilizada para a validação. Como discutido anteriormente, existem

Tabela 6. Resultados da validação de três serviços no conjunto de dados C

Serviço	Measure	kNN	SVM	NB	RF
Cassandra	Precisão	0,91	0.48	0.61	0.89
	Recall	0.89	0.35	0.51	0.75
	F1-score	0,90	0.33	0.50	0.76
propriedade rural	Precisão	0,92	0.27	0.56	0.71
	Recordação da	0.90	0.36	0.48	0.72
	F1-score	0,91	0.28	0.46	0.69
Broto	Precisão	0,88	0.31	0.47	0.85
	Relembrar	0.86	0.33	0.46	0.78
	F1-score	0,86	0.28	0.42	0.79

três contêineres que operam o serviço de homestead. Como uma aplicação de microserviço, a carga de trabalho e os dados de desempenho desses três contêineres devem ser simi- lar. Assim, o recipiente com a distância mais distante de outros será considerado como o recipiente anômalo. Um programa python é implementado para nos ajudar a diagar o recipiente anômalo, e ele obtém os últimos 20 dados de desempenho do InfluxDB, calcula a distância e mostra o resultado como mostrado na Fig. 4.

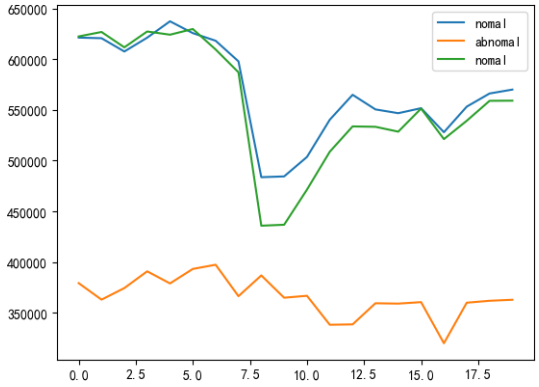

Fig. 4. Diagnóstico do recipiente anômalo.

6 Conclusão e trabalho futuro

Neste trabalho, analisamos a métrica de desempenho para microserivces baseados em recipientes, introduzimos duas fases para detectar anomalias com técnicas de aprendizagem de máquinas - niques, e então, propusemos um sistema de detecção de anomalias para microserivces baseados em recipientes. Nosso ADS se baseia nos dados de monitoramento de desempenho dos serviços

e recipientes, algoritmos de aprendizagem de máquinas para classificar comportamentos anômalos e nem-mal e o módulo de injeção de falhas para coleta de dados de desempenho em várias condições do sistema.

No futuro, será estudado um estudo de caso mais representativo na arquitetura de microserviços. Atualmente, o módulo de injeção de falha focou apenas em alguma falha específica de hardware e, no futuro, alguns cenários complicados de injeção podem ser adicionados a este módulo.

Referências

1. Vindeep Singh e et al. Arquitetura de microserviços baseada em contêineres para aplicações em nuvem. *Computação, Comunicação e Automação (ICCCA)*, 2017.
2. Carla Sauvanaud e et al. Detecção e diagnóstico de anomalias para serviços de nuvem: Experiências práticas e lições aprendidas. *Journal of Systems and Software*, 139:84-106, 2018.
3. Grzegorz Dwornicki Rusek, Marian e Arkadiusz Orowski. Um sistema descentralizado... tem para o balanceamento de carga de micro-serviços de contêineres na nuvem. *Conferência Internacional sobre Ciência de Sistemas. Springer, Cham, 2016.*, 2016.
4. Nano. Kratzke. Sobre microserviços, recipientes e seu impacto subestimado sobre o desempenho da rede. *arXiv preprint arXiv:1710.04049(2017).*, 2017.
5. A.; Kumar V. Chandola, V.; Banerjee. Detecção de anomalias: Uma pesquisa. *ACM Com- puting Surveys.*, 2009.
6. Tao Wang, Wenbo Zhang, Chunyang Ye, et al. Fd4c: Diagrama automático de falhas... estrutura nosis para aplicações web na computação em nuvem. *IEEE Transactions on Systems, Man, and Cybernetics (Transações IEEE em Sistemas, Homem e Cibernética): Sistemas*, 46(1):61-75, 2016.
7. Marcelo Amaral, Jorda Polo, et al. Avaliação do desempenho dos microserviços arquiteturas que utilizam recipientes. Em *Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on*, páginas 27-34. IEEE, 2015.
8. A. Ferreira W. Felter e et al. Uma comparação de desempenho atualizada de máquinas e recipientes de linux. *Relatório técnico RC25482(AUS1407-001), IBM*, 2014.
9. J. Kjallman R. Morabito e M. Komu. Hypervisors vs. virtualização leve: Uma comparação de desempenho. *IEEE International Conference on Cloud Engineering*, 2015.
10. Z. Zheng Y. Zhang e M.R. Lyu. Uma estrutura de previsão de desempenho on-line para sistemas orientados a serviços. *IEEE Transactions on Systems, Man, and Cybernetics*, 2014.
11. Haibo Mi, Huaimin Wang, et al. Rumo a granulação fina, sem supervisão, escalável diagnóstico de desempenho para sistemas de computação em nuvem de produção. *IEEE Transac- tions on Parallel and Distributed Systems*, 24(6):1245-1255, 2013.
12. Shigang Zhang, Krishna R Pattipati e et al. Diagnóstico dinâmico de falhas acopladas com atrasos de propagação e observação. *Transações IEEE em Sistemas, Homem e Cibernética: Systems*, 43(6):1424-1439, 2013.
13. Claus. Pahl. Containerização e a nuvem de paas. *IEEE Cloud Computing*, 2015.
14. T Warren Liao. Agrupamento de séries temporais - levantamento de dados. *Reconhecimento de padrões*, 38(11):1857-1874, 2005.
15. Yanping Chen, Keogh e et al. The ucr time series classification archive, julho 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.
16. Águas claras. Projeto Clearwater. http://www.projectclearwater.org/.