Amostragem Dinâmica para Telemetria em Microsserviços: Uma Abordagem Baseada em Aprendizado por Reforço e Entropia

Renan Martins Alves[1], **Jéferson Campos Nobre**[1], **Juliano Araujo Wickboldt**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brasil

{renan.alves, jcnobre, jwickboldt}@inf.ufrgs.br

Abstract. *This article proposes a RADAR agent (Reinforcement Learning Agent for Dynamic and Relevant Trace Sampling) that uses a reinforcement learning approach and data entropy assessment, ensuring more efficient capture of traces relevant to system monitoring. RADAR tests different sampling rules to discover which set is the most efficient. A test environment simulating a minimalist online store with several microservices distributed across a Kubernetes cluster served as the basis for the experiments, which evaluated the agent's convergence and system performance in relation to resource consumption and the quality of the collected data. The results demonstrated that RADAR was able to reduce network bandwidth consumption by 97.5% and CPU usage by 99% when compared to full data collection, also surpassing the efficiency of fixed sampling strategies. In addition to resource savings, the approach preserved the observability of critical scenarios, maintaining approximately 89% of rare trace patterns and increasing the average entropy of the stored information by approximately 25%, validating the feasibility of using entropy to orchestrate telemetry autonomously and efficiently.*

Resumo. *Este artigo propõe RADAR (Reinforcement learning Agent for Dynamic And Relevant trace sampling), um agente que utiliza aprendizado por reforço e avaliação de entropia para uma captura mais eficiente de traces relevantes para o monitoramento de um sistema. O RADAR testa diferentes regras de amostragem para descobrir qual o conjunto é o mais eficiente. Um ambiente de testes simulando uma loja online minimalista com diversos microsserviços distribuídos em um cluster Kubernetes foi a base para os experimentos, que avaliaram a convergência do agente e o desempenho do sistema em relação ao consumo de recursos e a qualidade dos dados coletados. Os resultados demonstraram que o RADAR foi capaz de reduzir o consumo de banda de rede em 97,5% e o uso de CPU em 99% quando comparado à coleta integral, superando também a eficiência de estratégias de amostragem fixa. Além da economia de recursos, a abordagem preservou a observabilidade de cenários críticos, mantendo cerca de 89% dos padrões de traces raros e aumentando a entropia média das informações armazenadas em aproximadamente 25%, validando a viabilidade de utilizar entropia para orquestrar telemetria de forma autônoma e eficiente.*

1. Introdução

A transição de arquiteturas monolíticas para microsserviços permitiu que organizações aumentassem a agilidade no desenvolvimento e a escalabilidade de suas aplicações [Kratzke 2018]. Entretanto, essa transição aumentou a complexidade da realização de tarefas de observabilidade de sistemas [Zhou et al. 2018]. Em ambientes que utilizam microsserviços, uma única requisição de usuário pode transitar por dezenas de serviços independentes, tornando o *tracing* distribuído uma ferramenta indispensável para compreender o fluxo de execução e diagnosticar falhas latentes [Zhou et al. 2018].

Apesar de sua importância, a coleta integral de *traces* em sistemas de larga escala é inviável. O volume de dados gerados pode sobrecarregar a largura de banda da rede, consumir ciclos excessivos de CPU para processamento e resultar em custos de armazenamento proibitivos. Para mitigar esse problema, estratégias de amostragem (*sampling*) são aplicadas. Métodos tradicionais, como o *head-sampling* probabilístico, decidem sobre o descarte de dados no início da transição, falhando em capturar eventos raros ou erros que só se manifestam no final do fluxo. Por outro lado, o *tail-sampling* permite decisões mais informadas, mas sua configuração manual é rígida e incapaz de se adaptar às variações dinâmicas de carga e comportamento típicas de ambientes em nuvem [Gomez Blanco 2023]. Soluções robustas como o Hindsight [Zhang et al. 2023], que gera e mantém todos os traces localmente, apenas enviando para o coletor após alguma anomalia ser detectada, ou o trabalho de Las-Casas *et.al.* [Las-Casas et al. 2018], que aglomera traces semelhantes e comuns em nodos de uma árvore e priorizam o descarte deles frente a necessidade de liberação de recursos ainda deixam lacunas visíveis, como a rigidez na definição de gatilhos para definir anomalias ou possuem falta de compatibilidade com as ferramentas mais modernas de observabilidade.

Neste cenário, surge a necessidade de mecanismos de observabilidade que possam selecionar, de forma autônoma, os dados mais relevantes. Este trabalho propõe o **RADAR** (*Reinforcement learning Agent for Dynamic And Relevant tracing sampling*), um *framework* que utiliza Aprendizado por Reforço (RL) para ajustar dinamicamente as políticas de amostragem em coletores OpenTelemetry. Este trabalho utiliza a **Entropia de Shannon** como métrica de recompensa, permitindo que o sistema priorize *traces* que apresentam maior "surpresa" ou diversidade informacional, descartando requisições redundantes que não agregam valor ao diagnóstico.

O objetivo principal deste artigo é apresentar uma arquitetura de amostragem dinâmica que equilibre o custo operacional e a qualidade da informação. As principais contribuições deste trabalho incluem:

- **Framework RADAR**: projeto e implementação prática de um agente de RL integrado ao ecossistema OpenTelemetry.
- **Métrica Baseada em Entropia**: Uma metodologia para quantificar a relevância de *traces* distribuídos sem a necessidade de rotulagem prévia de dados.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta o referencial teórico necessário para o entendimento desta proposta, enquanto a Seção 3 traz um panorama dos trabalhos relacionados. A Seção 4 descreve a metodologia proposta. A Seção 5 apresenta os resultados obtidos, enquanto a Seção 6 discute de forma mais aprofundada os principais resultados deste trabalho. Finalmente, a Seção 7 apresenta considerações finais e os próximos passos para o aprimoramento do projeto.

2. Referencial Teórico

Esta seção apresenta o referencial teórico utilizado para o desenvolvimento do artigo, incluindo os conceitos de observabilidade em ambientes de microsserviços e de aprendizado por reforço utilizados na modelagem do problema.

2.1. Observabilidade

A observabilidade permite inferir o estado interno de sistemas complexos a partir de dados externos, superando limitações de métricas e logs tradicionais em microsserviços altamente dinâmicos [Majors et al. 2022, Zhou et al. 2018]. Para resolver a dificuldade de replicar erros, é fundamental a propagação de contexto e a correlação automática de eventos [Gomez Blanco 2023]. O sistema deve ser instrumentado nativamente para emitir traces, logs e métricas, garantindo as informações necessárias para diagnóstico sem intervenções posteriores [Majors et al. 2022].

O OpenTelemetry se consolida como uma estrutura padronizada para a coleta de dados de sistemas complexos [Majors et al. 2022]. Seus componentes centrais são os traces (registro do caminho completo percorrido por uma requisição) e spans (unidade de trabalho individual). Essa estrutura hierárquica fornece o contexto detalhado de cada etapa de execução em um sistema distribuído [Majors et al. 2022].

Para otimizar custos de processamento e armazenamento sem comprometer a observabilidade, utilizam-se técnicas de amostragem [Gomez Blanco 2023]. A abordagem de head-sampling decide a coleta no início do processo, de forma simples e com baixo custo, mas sem o contexto de toda a requisição. Já o tail-sampling ocorre após a conclusão do trace, permitindo critérios granulares e sofisticados, garantindo que dados críticos sejam preservados [Majors et al. 2022].

2.2. Aprendizado por Reforço

Aprendizado por reforço (Reinforcement Learning, RL) é uma abordagem em que um agente aprende a tomar decisões por meio de tentativa e erro, interagindo com um ambiente e recebendo sinais de recompensa em resposta às suas ações. Diferentemente do aprendizado supervisionado, no RL o agente recebe apenas feedback escalar (recompensa) e precisa descobrir uma política de decisão que maximize o retorno acumulado ao longo do tempo. Esse esquema explicita o papel central dos três componentes fundamentais do RL: o agente (quem decide), o ambiente (com o qual ele interage) e a recompensa (sinal que orienta o aprendizado) [Sutton and Barto 2018].

O agente é a entidade que implementa a política de decisão e realiza o mapeamento das observações para ações. Ele mantém estimativas internas, atualizando-as com base na experiência coletada. Em termos computacionais, é o algoritmo responsável por balancear exploration (tentar novas ações para obter mais informação) e exploitation (aproveitar o conhecimento já adquirido). O ambiente é tudo aquilo que está "do lado de fora" do agente e que responde às suas ações com novas observações e recompensas. Podendo representar um sistema físico ou computacional complexo, a ideia central é que o ambiente define a dinâmica e as consequências das ações do agente. A recompensa é o sinal escalar fornecido pelo ambiente que representa o objetivo do problema, especificado como a maximização da soma de recompensas recebidas ao longo do tempo. A escolha dessa função é crítica: recompensas mal definidas podem levar o agente a comportamentos indesejados [Sutton and Barto 2018].

3. Trabalhos Relacionados

Las-Casas *et.al.* propõem aumentar a probabilidade de coleta de *traces* únicos por meio do cálculo de distância euclidiana entre *traces*, representados como vetores numéricos derivados de grafos de eventos [Las-Casas et al. 2018]. A partir dessa métrica, aplica-se uma clusterização hierárquica que organiza *traces* similares em uma árvore, com eventos frequentes formando árvores profundas e *traces* raros permanecendo próximos a raiz. A seleção de amostragem ocorre com o algoritmo caminhando pela árvore de forma aleatória até encontrar uma folha, favorecendo a seleção de *traces* raros, próximos da raiz. Embora explore a diversidade estrutural dos dados, a abordagem depende de clusterização estática e representação vetorial específica. Em contraste, este trabalho propõe o uso de aprendizado por reforço para adaptar dinamicamente a política de amostragem, operando diretamente sobre o OpenTelemetry e sem conversões complexas.

Já Pogshoyan *et.al.* utilizam inicialmente uma combinação de *head* e *tail sampling* para diminuir a quantidade de dados inicial a ser passada pela solução proposta [Poghosyan et al. 2024]. Depois, passa por uma etapa de redução de ruído, onde *traces* raros são descartados e comuns preservados, uma vez que erros recorrentes possuem maior chance de explicar uma falha sistêmica. Os dados são transformados em formato tabular e inseridos em um sistema de aprendizado de máquina (ML), como o *RIPPER*. O sistema então gera regras simples que apontam para condições que geram falhas, e cada regra utiliza a Teoria *Dempster-Shafer* para medir a incerteza da regra. Em contraste com a etapa de redução de ruído deste método, que descarta ativamente eventos raros, a proposta deste trabalho busca maximizar a entropia justamente para capturar casos raros e anômalos. Além disso,a geração estática de regras é substituída por um ciclo de *feedback* contínuo guiado por recompensa, eliminando a necessidade de converter *traces* hierárquicos em tabelas planas para o processamento.

Luo *et.al.* propõem um sistema de *logging* proporcional à culpa, no qual gatilhos leves detectam a primeira ocorrência de um problema e, a partir disso, um algoritmo atribui um ranking de culpa aos métodos da aplicação, estimando sua relevância para a causa raiz [Luo et al. 2018]. Quando um problema é identificado, o sistema ativa *logging* intensivo apenas nos métodos mais prováveis de serem responsáveis, utilizando monitoramento contínuo de baixo custo, análise do grafo de chamadas da requisição e critérios específicos para exceções ou problemas de desempenho. Essa estratégia concentra a coleta agressiva de dados apenas onde é necessário. Em contraste, enquanto essa abordagem depende de gatilhos declarativos definidos manualmente e de instrumentação dinâmica em tempo de execução, o presente trabalho propõe uma solução parcialmente autônoma, baseada em múltiplas regras e no uso da entropia como sinal universal de interesse, atuando apenas na configuração de amostragem do coletor de telemetria.

Zhang et al. propõem o Hindsight, um sistema de tracing distribuído voltado à captura eficiente de edge cases em ambientes de alta escala, baseado em amostragem retroativa: todos os spans são gerados localmente, mas apenas coletados quando gatilhos programáveis detectam anomalias, como alta latência ou erros [Zhang et al. 2023]. Nesta abordagem, os agentes armazenam temporariamente os spans em memória e, ao identificar um problema, o sistema recupera retroativamente os dados da requisição afetada, desacoplando a geração da ingestão dos traces. Os autores demonstram que o método captura a maioria dos casos raros com impacto mínimo no desempenho, operando com

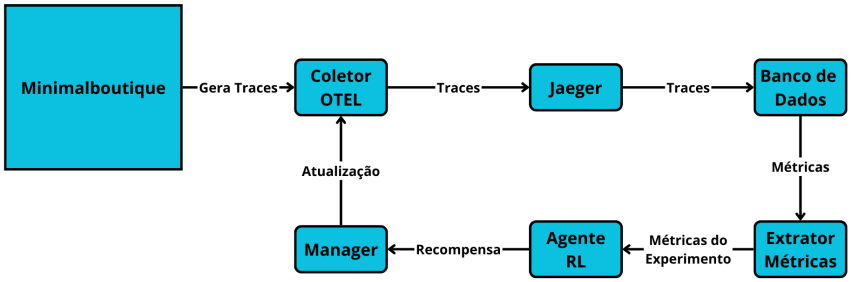

Figura 1. Arquitetura do RADAR

latência na ordem de nanossegundos, além de ser compatível com APIs como OpenTelemetry e X-Trace. Contudo, o Hindsight atua de forma reativa, dependendo de gatilhos pré-definidos, enquanto o presente trabalho avança ao empregar RL para identificar proativamente traces relevantes por meio da maximização da entropia, permitindo capturar cenários que não seriam detectados por gatilhos estáticos convencionais.

4. Metodologia para Amostragem Adaptativa de Dados de Telemetria

Esta seção descreve a abordagem metodológica adotada para o desenvolvimento e implementação do sistema **RADAR** (*Reinforcement learning Agent for Dynamic And Relevant trace sampling*). A metodologia está estruturada em quatro pilares fundamentais: (1) a fundamentação em padrões de observabilidade; (2) a formalização do problema de amostragem como um ambiente de RL; (3) a utilização da entropia de Shannon como métrica de relevância informacional; e (4) a implementação de um ciclo de retroalimentação automatizado em infraestrutura conteinerizada. O objetivo central é permitir que o sistema aprenda, de forma autônoma, quais conjuntos de regras de amostragem preservam dados de maior utilidade diagnóstica, reduzindo o custo de observabilidade.

4.1. Arquitetura do sistema RADAR

A arquitetura proposta baseia-se em um ciclo de retroalimentação fechado que integra o monitoramento da aplicação com a tomada de decisão autônoma. O sistema é estruturado em três componentes principais que operam ciclicamente.

Inicialmente, o **Ambiente Observado** engloba a aplicação de microsserviços e o coletor de OTel, onde as regras são aplicadas em tempo real. Em seguida, o **Agente de Controle** atua como o motor de decisão, selecionando subconjuntos de políticas de amostragem a partir de um catálogo pré-definido para satisfazer a função de recompensa. Por fim, o **Módulo de Análise de Métricas** processa os *traces* exportados para extrair características estatísticas, calculando a entropia e o volume de dados, que retroalimentam o agente para o próximo ciclo de aprendizado, como mostrado na figura 1.

4.2. Modelagem do Aprendizado por Reforço

Como o RADAR atua na configuração global do coletor de telemetria sem observar o estado prévio do ambiente (como uso de CPU, memória ou volume de tráfego no instante exato) antes da tomada de decisão, o desafio da amostragem adaptativa não configura um Processo de Decisão de Markov (MDP) tradicional, pois não há restrição de estados observável pelo agente. Em vez disso, o problema é formalizado como um **Multi-Armed Bandit combinatório**.

Para resolver este problema, utilizamos uma adaptação do algoritmo de gradiente de política **REINFORCE** para um cenário *stateless*, onde o agente aprende a melhor distribuição de probabilidades para selecionar combinações de regras com base unicamente no histórico de recompensas obtidas. O problema é definido pelos seguintes componentes fundamentais: o espaço de ações (A) e a função recompensa (R).

O espaço de ações (A) é discreto e combinatório. O RADAR possui um catálogo pré-definido composto por N políticas de amostragem independentes. A cada episódio t, a ação do agente $a_t \in \{1, 0\}^N$ consiste na seleção de um vetor binário. Cada posição do vetor i indica a ativação ($a_{t,i} = 1$) ou desativação ($a_{t,i} = 0$) da política correspondente na configuração do coletor OpenTelemetry. A probabilidade de cada regra ser selecionada é mantida internamente pelo agente e atualizada via gradiente ascendente.

A função de recompensa (R) é o núcleo dessa metodologia, projetada para equilibrar a riqueza da informação com o custo operacional. Ela é formalizada pela equação:

$$R = \pi.H(X) - \beta.P(n) \tag{1}$$

Onde $H(X)$ representa a entropia da informação, medindo a diversidade dos *traces* coletados, e $P(n)$ é uma penalidade logística aplicada ao volume de dados (n). Através do gradiente ascendente, o agente aumenta a probabilidade de selecionar combinações de regras que resultem em alta entropia com baixo volume. π e β são os dois hiperparâmetros de ponderação, que controlam o *trade-off* entre obter mais diversidade de informação e economizar recursos computacionais. Ajustes nesses valores permitem adaptar o RADAR para ambientes mais permissivos com custos (aumentando π) ou ambientes altamente restritos (aumentando β).

4.3. Relevância Informacional via Entropia

Para quantificar a utilidade dos dados sem supervisão humana, utiliza-se a **Entropia de Shannon**. No contexto de telemetria, a entropia permite medir a "surpresa" ou variabilidade dos caminhos de execução. para o cálculo, os traces (estruturas complexas de árvores de spans) são convertidos em representações textuais determinísticas.

Este processo de conversão envolve a ordenação hierárquica dos *spans* e a filtragem de atributos de alta cardinalidade irrelevantes (como endereços IP dinâmicos). Adicionalmente, valores contínuos como a latência são discretizados em intervalos (*buckets*) para evitar que variações naturais de rede inflem artificialmente a métrica. A entropia final é calculada sobre a distribuição de frequência desses *traces* únicos, permitindo ao agente priorizar comportamentos anômalos.

4.4. Orquestração de Ciclo e Feedback

A implementação do RADAR utiliza *scripts* em Python para gerenciar o ciclo da vida do experimento. No início de cada episódio, é necessária a substituição do arquivo atual de configuração do coletor OpenTelemetry. Para isso, o sistema gera dinamicamente um novo arquivo de configuração YAML e substitui o ConfigMap do ambiente Kubernetes, acrescido de uma hash de identificação que é adicionada como uma anotação.

Para aplicar as novas políticas sem interromper a telemetria, o sistema utiliza a API do Kubernetes para orquestrar uma atualização gradual (*rolling update)*. Após a atu-

alização do ConfigMap, o *script* aplica um *patch* no *deployment* do coletor, inserindo uma anotação de metadados com o *hash* da nova configuração. Isso ativa o gatilho para que o controlador do Kubernetes inicie novos *pods* com a configuração atualizada de forma transparente, garantindo que o agente comece a medir o ambiente somente quando o novo estado estiver estável. A tabela 1 demonstra o fluxo do ciclo de vida de um episódio do RADAR. A implementação do RADAR está disponível publicamente no seguinte repositório: `https://github.com/ComputerNetworks-UFRGS/RADAR`.

Tabela 1. Fluxo de Implementação: Ciclo de Vida do RADAR

Etapa	Componente	Ação Realizada
1. Inicialização	Script Python	Gera o arquivo YAML dinâmico + Hash.
2. Persistência	K8s ConfigMap	Atualiza o ConfigMap com a nova política do OTel.
3. Gatilho (Trigger)	K8s API (Patch)	Atualiza o Deployment injetando o Hash nos metadados.
4. Orquestração	K8s Controller	Identifica a mudança e inicia o *Rolling Update*.
5. Estabilização	Novos Pods OTel	Coletor sobe e mede o ambiente após o *Ready State*.

5. Avaliação

Esta seção apresenta a avaliação experimental do sistema RADAR, mostrando o ambiente de testes, a capacidade de convergência do modelo, o consumo de recursos e a eficácia na preservação de informações raras.

5.1. Ambiente Experimental

A validação do RADAR ocorre no **MinimalBoutique**[1], um ambiente de microsserviços distribuído em um cluster Kubernetes que simula uma loja online. O MinimalBoutique foi desenvolvido internamente por este grupo de pesquisa especificamente para possuir controle total sobre o código-fonte e a topologia dos microsserviços, sendo possível criar um ambiente de base isolado, garantindo que as anomalias detectadas pelo RADAR fossem exclusivamente derivadas dos cenários produzidos pelo experimento, sem a interferência de ruídos intrínsecos a *benchmarks* de terceiros. A aplicação utiliza o padrão *Database-per-service* para forçar a comunicação via API e maximizar a geração de *spans* de rede, essenciais para testar a captura de latências distribuídas. A figura 2 mostra a arquitetura da loja online.

Para simular o tráfego real, utiliza-se o **Locust**[2] configurado para gerar uma carga estocástica. O comportamento dos usuários virtuais é modelado como uma Cadeia de Markov, transitando entre navegação, adição ao carrinho e checkout com probabilidades variadas. Essa estrutura garante uma mistura heterogênea de telemetria: Uma grande quantidade de traces simples e repetitivos e poucos traces longos e complexos, desafiando a capacidade de seleção inteligente do agente.

[1] O MinimalBoutique está disponível publicamente no repositório: `https://github.com/ComputerNetworks-UFRGS/minimal-boutique`

[2] Locust: An open source load testing tool: `https://locust.io/`

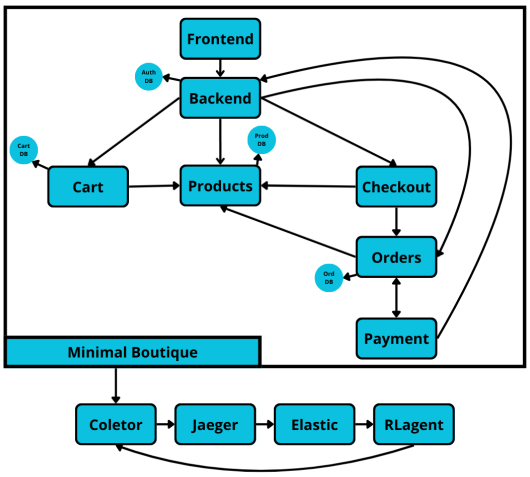

Figura 2. Arquitetura do Minimalboutique

5.2. Experimentos

Para avaliar a eficiência do método conforme o número de regras disponíveis para a ação do agente aumenta, foram realizados quatro experimentos incrementais. As regras foram agrupadas em três conjuntos: regras estruturais do sistema (latência e *spans*), regras probabilísticas e regras específicas do ambiente, como demonstrado na tabela 2.

Tabela 2. Regras selecionadas por experimento (formato incremental)

Experimento	Regras selecionadas	Total
1	Latência $>$ 500 ms; Probabilidade de 10%; Compras $>$ R$ 500	3
2	Regras do experimento anterior + Traces com mais de 20 spans; Probabilidade de 20%; Pagamentos recusados	6
3	Regras do experimento anterior + Consultas lentas ao banco de dados; 1% de todas as alterações do sistema; Mais de 10 itens no carrinho	9
4	Regras do experimento anterior + Erros; 1% de todas as operações no serviço de produtos; Mais de 4 pedidos realizados	12

Os resultados demonstram que, independentemente da quantidade de regras disponíveis, o RADAR foi capaz de convergir para uma política estável, maximizando a recompensa recebida ao longo dos episódios. Observou-se que a entropia foi maximizada e convergiu para um valor mais alto do que no início do experimento. Enquanto o experimento com três regras limitou de maneira mais brusca a quantidade de traces por possuir menos opções, o cenário com doze regras permitiu atingir níveis maiores de riqueza dos dados coletados ao custo de um tempo maior para convergir, como pode ser visto na figura 3, que demonstra a convergência das recompensas e na figura 4, que demonstra a convergência da entropia dos dados. Vale ressaltar que, para melhor visualização dos gráficos, optou-se por apresentar apenas os casos mais extremos (com 3 e 12 regras) nas figuras 3 e 4. Destacamos que os cenários com 6 e 9 regras foram testados e os resultados

apresentam as mesmas tendências observadas.

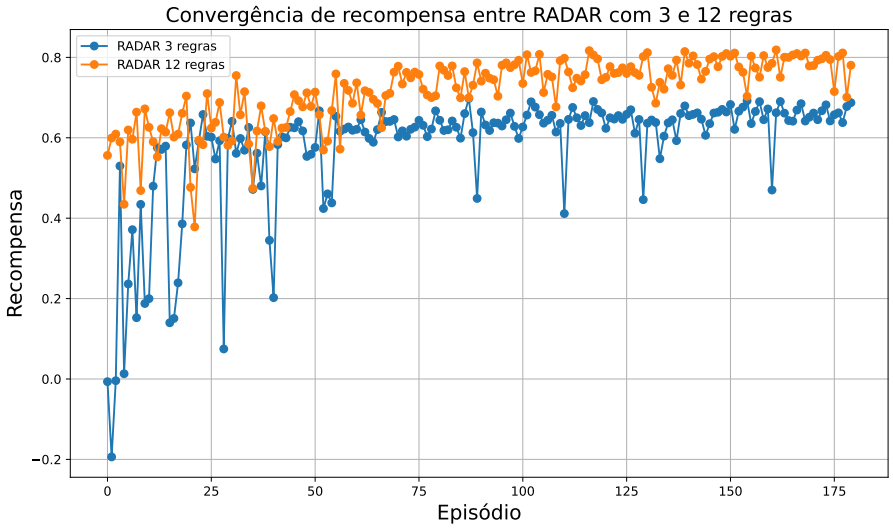

Figura 3. Convergência das recompensas com 3 e 12 regras

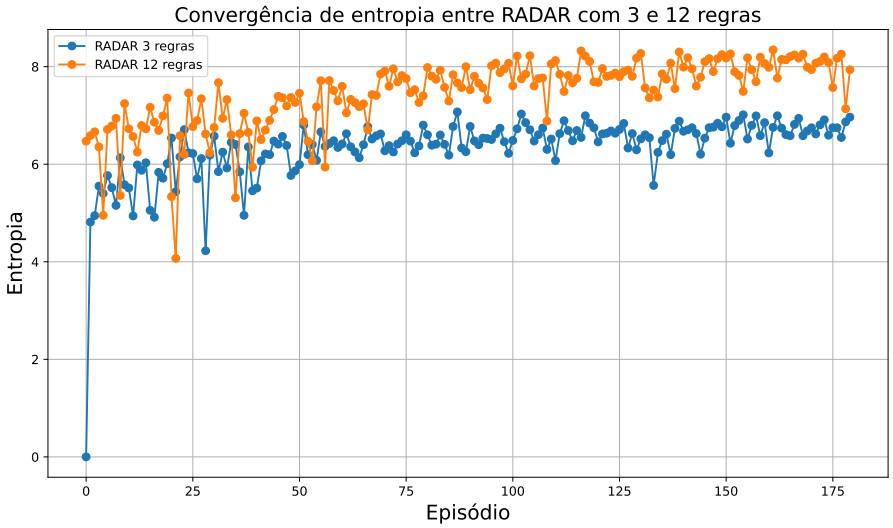

Figura 4. Convergência da entropia com 3 e 12 regras

5.3. Uso de Recursos Computacionais e de Comunicação

Para os testes de desempenho, foram executados por cinco horas em três coletores diferentes: Coletor armazenando 100% dos traces: adicionada uma única regra que coleta 100% dos traces que chegam ao coletor; coletor armazenando 20% dos traces: adicionada uma única regra que coleta 20% dos traces que chegam ao coletor. Esse valor foi o *midpoint* utilizado para o cálculo de penalidade pelo número de traces, e é o valor máximo que o RADAR poderia atingir sem começar a sofrer grandes penalizações; RADAR: coletor utilizando o nosso agente de aprendizado por reforço para selecionar as melhores políticas para o ambiente. Na figura 5, temos no eixo x o tempo em minutos, e no eixo Y a taxa de transmissão de banda de saída do coletor, com o gráfico representando a média gasta ao longo do tempo.

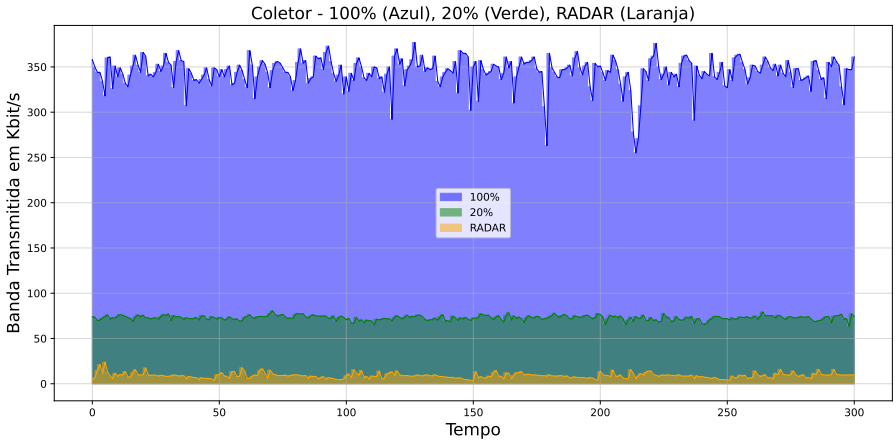

Figura 5. Taxa de transmissão de banda com 100%, 20% e com RADAR

Os resultados demonstram que o RADAR diminuiu significativamente a taxa de transmissão comparada com o coletor de 100% que possui uma média de 344 Kbits/s, enquanto o RADAR possui uma média de 9 Kbits/s, uma redução de 97,5%. Já comparado com o coletor com 20% de todas as regras, que possui uma média de 73 Kbits/s, a redução foi de 87,5%. Considerando o coletor de 100% como a transmissão máxima, o RADAR possui a transmissão equivalente a um coletor com uma regra de coleta de 2,5%, sem a parte probabilística de coleta de erros.

Na figura 6, de forma similar, apresentamos o percentual de CPU utilizada por cada coletor, com o gráfico representando a CPU média utilizada por cada solução ao longo do tempo. Os resultados mostram que com o uso de CPU médio do coletor 100% sendo de 1,99, e o uso médio do RADAR sendo 0,02, uma diminuição de 99%. Já para o coletor de 20%, a média de uso foi de 0,38, uma redução de 95% comparado com o RADAR. Isso se explica pelo descarte de *traces* que não são processados e enviados para o Jaeger.

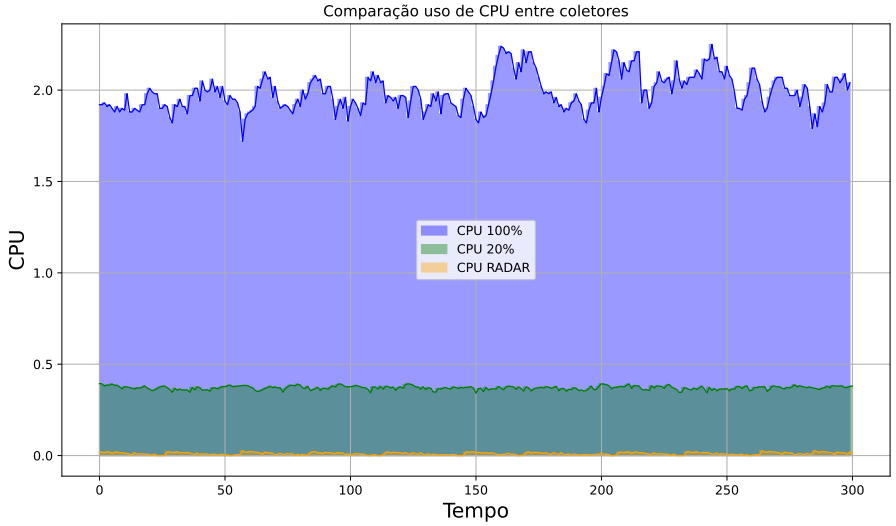

Figura 6. Uso de CPU com 100%, 20% e com RADAR

De forma similar, o gráfico da figura 7 apresenta a quantidade de memória utilizada por cada coletor em MB ao longo do tempo. Os resultados mostram que a média do coletor com 100% dos traces foi de 476 MB, o que comparado aos 200 MB do RADAR, representa uma diminuição de 58% de uso de memória. Mesmo com o coletor a 20%, onde o uso médio foi de 372 MB, houve uma redução de 46%.

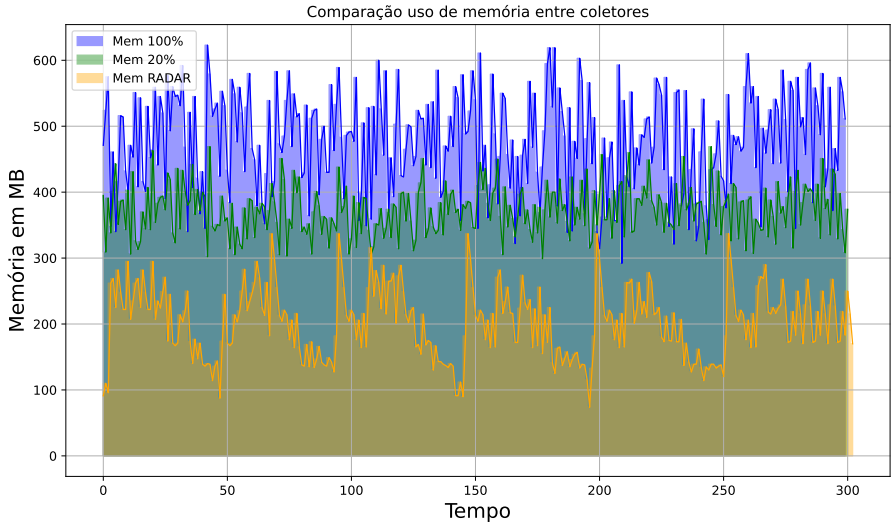

Figura 7. Uso de Memória com 100%, 20% e com RADAR

5.4. Análise Granular de Traces

Para testarmos se os *traces* raros eram efetivamente capturados pelo RADAR, foi implementado um coletor OTel com dois *pipelines* de processamento independentes. O primeiro *pipeline* utilizou o conjunto de regras selecionados pelo RADAR como a configuração ótima, enquanto o segundo foi configurado com uma política de amostragem que coleta 100% dos *traces*. Com o objetivo de permitir a distinção entre os dois fluxos de dados durante a análise, foi adicionada uma *key* identificador dos *traces* produzidos por cada *pipeline*: ***baseline*** para a coleta integral e ***experiment*** para o configurado com as regras do RADAR.

Após o período de coleta, os *traces* provenientes de ambos os *pipelines* foram submetidos às funções de classificação e normalização do RADAR, que inclui a transformação dos *traces* em representações textuais canônicas, com a remoção de atributos únicos e variáveis, a fim de reduzir ruído e permitir comparações estruturais entre as execuções. Em seguida, foi realizado o ranqueamento dos padrões dos *traces* menos frequentes a partir do conjunto de dados do *pipeline baseline*. Por fim verificou-se a presença desses padrões raros no conjunto coletado pelo pipeline experimental para o cálculo da porcentagem de cobertura.

O experimento foi realizado em dez casos de teste, durante uma hora cada, com o pipeline duplo. Os *traces* considerados raros são aqueles que aparecem uma única vez em toda a execução do teste. A média dos *traces* coletados é mostrada na tabela abaixo:

Inicialmente, é possível ver que a porcentagem de padrões únicos no *baseline* em relação ao número total de traces é de 12,96%, enquanto com as regras criadas selecionadas pelo RADAR é de 94%. Ao calcular a porcentagem dos traces únicos do *baseline*

Tabela 3. Resumo de traces

Tipo	Nº de traces	Nº de padrões únicos	% Presente
baseline	484967	63237	89,16%
experiment	61122	57460	85,57%

que estão presentes nos traces raros coletados pelo RADAR, a porcentagem obtida foi de 89,16%, um valor alto quando comparado com a diminuição da quantidade de traces coletados e o número de padrões únicos coletados. Os resultados reforçam o potencial do RADAR como solução prática e de baixo acoplamento para observabilidade adaptativa em sistemas distribuídos modernos.

6. Discussão

Os resultados obtidos confirmam que a observabilidade em arquiteturas de microsserviços apresenta um dilema fundamental: a necessidade de coletar dados detalhados versus o custo proibitivo de processar o imenso volume de telemetria gerado. A abordagem do RADAR pode ser uma estratégia viável para equilibrar esse dilema ao utilizar a entropia como sinal universal de interesse.

6.1. Eficiência e Equilíbrio de Recursos

A redução de 97,5% no consumo de banda e de 99% do uso de CPU demonstra que o descarte inteligente de dados redundantes é mais efetivo do que as abordagens tradicionais de amostragem probabilística (*head-sampling*). Enquanto o coletor de 100% gera um volume massivo de dados repetitivos que refletem operações bem sucedidas, o RADAR aprendeu a identificar e priorizar os caminhos de execução que realmente agregam valor informacional.

Comparado à amostragem fixa em 20%, o RADAR não apenas consumiu menos recursos, mas o fez de forma a maximizar a recompensa, o que indica que a economia de processamento não ocorreu à custa da perda aleatória de informação, mas sim por meio de uma filtragem qualitativa. Isso demonstra a possibilidade de usar uma função recompensa multiobjetivo, que penaliza o volume excessivo enquanto premia a diversidade estrutural dos *traces*.

6.2. Preservação da Observabilidade e Valor da Informação

A manutenção de cerca de 89% dos padrões de *traces* raros é um dos resultados mais significativos, pois eventos raros são justamente os mais relevantes para diagnósticos e detecção de anomalias. Ao aumentar a entropia média das informações armazenadas em aproximadamente 25%, o RADAR comprovou sua capacidade de distinguir entre *traces* repetitivos e redundantes e aqueles que são anômalos.

Diferente de sistemas como o **Hindsight**, que operam de maneira reativa e dependem de gatilhos de anomalia pré-definidos, o RADAR atua de forma proativa. O agente descobriu proativamente quais *traces* eram relevantes através da maximização da entropia, permitindo a captura de cenários de interesse que não ativariam um gatilho estático convencional. Além disso, ao operar inteiramente em espaço de usuário através de OpenTelemetry, o Radar garante maior portabilidade e evita a complexidade da injeção dinâmica de código ou modificações invasivas no *kernel*.

6.3. Viabilidade de Orquestração Autônoma

A convergência estável da política, mesmo com um número crescente de regras, reforça o potencial do RADAR como uma solução prática e de baixo acoplamento para sistemas distribuídos modernos. A metodologia demonstrou que é possível reduzir drasticamente o volume de dados sem sacrificar a observabilidade de cenários críticos, validando a viabilidade de utilizar entropia para orquestrar telemetria de forma autônoma e eficiente. Conclui-se que a aplicação de agentes inteligentes baseados em entropia oferece um equilíbrio dinâmico superior às configurações manuais de *tail-sampling*, que são difíceis de manter em ambientes dinâmicos de nuvem.

6.4. Ameaças à validade e Generalização

Apesar dos resultados promissores na redução de recursos e manutenção da entropia, este estudo apresenta limitações em relação à sua validade externa. A avaliação experimental foi conduzida exclusivamente no MinimalBoutique, um ambiente construído pelo próprio grupo. Embora o gerador de carga simule um ambiente e-commerce real, a arquitetura e os padrões de comunicação refletem um domínio único de aplicação. Consequentemente, a estabilidade de convergência do agente e a configuração ideal dos hiperparâmetros podem variar quando submetidas a topologias mais complexas ou *workloads* não previstos. Portante, a generalização do modelo empírico do RADAR para sistemas de produção heterogêneos ainda requer validação em *benchmarks* independentes.

7. Considerações Finais

A observabilidade em microsserviços enfrenta o dilema entre o detalhamento diagnóstico e o custo proibitivo da telemetria. Abordagens tradicionais mostram-se ineficientes ao descartar eventos críticos e depender de configurações rígidas incapazes de se adaptar a ambientes dinâmicos. Este trabalho propõe o **RADAR**, um *framework* autônomo que integra aprendizado por reforço e a Entropia de Shannon para ajustar dinamicamente as políticas de amostragem. Ao utilizar a entropia como métrica de diversidade, o sistema prioriza traces anômalos e reduz a redundância sem a necessidade de intervenção direta do usuário.

Experimentos comprovaram a eficiência da proposta: o RADAR reduziu o consumo de banda em 97,5% e o uso de CPU em 99% em comparação à coleta integral. A qualidade informacional foi preservada, mantendo 89% dos padrões de *traces* raros e elevando a entropia média dos dados em 25%. Tais resultados validam a hipótese de que a orquestração autônoma baseada em entropia oferece um equilíbrio dinâmico superior às estratégias convencionais, garantindo alta observabilidade com custo computacional drasticamente reduzido.

Apesar dos resultados promissores, a abordagem apresenta oportunidades para expansão e refinamento. Sugere-se para trabalhos futuros:

- **Evolução do descarte de regras**: Substituir o descarte total de regras pela porcentagem probabilística de sua ativação, possibilitando descobrir a porcentagem probabilística correta que permita otimizar a entropia da regra em si, enquanto diminui seus *traces* coletados.
- **Contextualização do estado**: Expandir a modelagem do problema para considerar o estado do cluster (ex: uso atual de CPU, horário do dia) na tomada de decisão, transformando o problema de *Bandit* em um MDP completo.

- **Integração com métricas de negócios**: Incorporar métricas de impacto do negócio na função recompensa, alinhando a estratégia de amostragem não apenas a diversidade técnica, mas à prioridade do negócio
- **Validação em Benchmarks Independentes**: Expandir a avaliação experimental do RADAR para aplicações de referência da indústria e da academia, a fim de testar a generalização da abordagem em cenários com maior densidade de serviços, diferentes linguagens de programação e *workloads* de estresse independentes.

Agradecimentos

O presente trabalho foi apoiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – Código de Financiamento 001 – e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPQ), Bolsas PQ processo nº 308075/2025-0 e 315427/2023-0.

Referências

Gomez Blanco, D. (2023). *Practical OpenTelemetry: Adopting Open Observability Standards Across Your Organization*. Apress, Berkeley, CA.

Kratzke, N. (2018). A brief history of cloud application architectures. *Applied Sciences*, 8(8):1368.

Las-Casas, P., Mace, J., Guedes, D., and Fonseca, R. (2018). Weighted sampling of execution traces: Capturing more needles and less hay. In *ACM Symposium on Cloud Computing*, SoCC '18, page 326–332, New York, NY, USA.

Luo, L., Nath, S., Sivalingam, L. R., Musuvathi, M., and Ceze, L. (2018). Troubleshooting transiently-recurring problems in production systems with blame-proportional logging. In *2018 Usenix Annual Technical Conference*, USENIX ATC '18, page 321–334, USA.

Majors, C., Fong-Jones, L., and Miranda, G. (2022). *Observability Engineering*. O'Reilly Media.

Poghosyan, A., Harutyunyan, A., Davtyan, E., Petrosyan, K., and Baloian, N. (2024). The diagnosis-effective sampling of application traces. *Applied Sciences*, 14(13).

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition.

Zhang, L., Xie, Z., Anand, V., Vigfusson, Y., and Mace, J. (2023). The benefit of hindsight: Tracing {Edge-Cases} in distributed systems. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 321–339.

Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Li, W., and Ding, D. (2018). Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering*, 47(2):243–260.