UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

Pedro Correia de Carvalho de Queiroz Lima

Avaliação Experimental da Resiliência do Celery em Cenários de Falhas

Recife

2025

Pedro Correia de Carvalho de Queiroz Lima

Avaliação Experimental da Resiliência do Celery em Cenários de Falhas

Trabalho apresentado ao Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para conclusão do Curso de Ciência da Computação.

Orientador (a): Breno Miranda

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Lima, Pedro Correia de Carvalho de Queiroz.
Avaliação experimental da resiliência do Celery em cenários de falhas / Pedro Correia de Carvalho de Queiroz Lima. - Recife, 2025.
35 p. : il.

Orientador(a): Breno Miranda
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2025.
Inclui referências.

1. Resiliência de sistemas. 2. Injeção de falhas. 3. Celery. 4. Tarefas assíncronas. I. Miranda, Breno. (Orientação). II. Título.

000 CDD (22.ed.)

PEDRO CORREIA DE CARVALHO DE QUEIROZ LIMA

Avaliação experimental da resiliência do Celery em cenários de falhas

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Aprovado em: 03/04/2025

BANCA EXAMINADORA

_____

Prof. Dr. Breno Alexandro Ferreira de Miranda (Orientador)

Universidade Federal de Pernambuco

_____

Profa. Dra. Paola Rodrigues de Godoy Accioly (Examinador Interno)

Universidade Federal de Pernambuco

AGRADECIMENTOS

Agradeço à minha família pelo apoio em todas as etapas da minha jornada, à minha companheira Clara por me inspirar a ser uma pessoa cada vez melhor, aos meus amigos pela amizade e motivação ao longo do caminho, e ao meu orientador Breno pelos conselhos valiosos durante o desenvolvimento deste trabalho.

RESUMO

A execução confiável de tarefas assíncronas é essencial para garantir eficiência e escalabilidade de sistemas distribuídos de grande escala, e o Celery se destaca nesse cenário como uma das ferramentas mais utilizadas em aplicações Python para o processamento de tarefas em segundo plano, oferecendo uma forma direta e prática de estruturar sistemas robustos. No entanto, sistemas de fila como o Celery frequentemente enfrentam gargalos de desempenho que só se manifestam sob cargas reais, e por isso, a prática de testes de resiliência surge como uma abordagem eficaz para antecipar possíveis problemas através da injeção de falhas e a análise das consequências geradas no comportamento do sistema. Ainda não existem ferramentas nativas para testes desse tipo no Celery, o que dificulta a compreensão de seu comportamento em ambientes críticos. Diante disso, este trabalho propõe uma avaliação experimental da resiliência do Celery por meio da injeção de falhas controladas, e os resultados obtidos indicam que, com o uso de configurações adequadas, é uma ferramenta capaz de manter a continuidade do processamento e apresentar um bom grau de resiliência mesmo sob alta demanda, demonstrando seu potencial como uma ferramenta robusta e confiável.

Palavras-chaves: Resiliência de Sistemas, Injeção de Falhas, Celery, Tarefas Assíncronas.

ABSTRACT

Reliable execution of asynchronous tasks is essential to ensure the efficiency and scalability of large-scale distributed systems, and Celery stands out in this context as one of the most widely used tools in Python applications for background task processing, offering a straightforward and practical way to structure robust systems. However, task queue systems like Celery often face performance bottlenecks that only emerge under real workloads. As a result, resilience testing emerges as an effective approach to anticipate potential issues through fault injection and analysis of the resulting system behavior. Currently, there are no native tools in Celery for this type of testing, which makes it difficult to understand its behavior in critical environments. In this scenario, this work proposes an experimental evaluation of Celery's resilience through the injection of controlled faults, and the results indicate that, with appropriate configurations, it's a tool capable of maintaining processing continuity and demonstrating a good level of resilience even under high demand, highlighting its potential as a robust and reliable tool.

Keywords: System Resilience, Fault Injection, Celery, Asynchronous Tasks.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Em sistemas de grande escala, a execução de tarefas assíncronas e distribuídas é fundamental para garantir a eficiência e escalabilidade das operações, principalmente as que exigem o processamento de grandes volumes de dados ou a execução de tarefas críticas. Para garantir isso, muitos sistemas adotam ferramentas de fila de tarefas que provêm um bom balanceamento de carga e processamento contínuo sem sobrecarregar os recursos do sistema (THALLAPALLY, 2025).

O Celery é uma das ferramentas open-source mais populares para a execução de tarefas distribuídas e assíncronas em Python, atualmente contando com mais de 25 mil estrelas no GitHub (Celery Project, 2025). Se destaca por permitir a execução de tarefas em segundo plano de uma forma prática e escalável, sendo uma tecnologia amplamente utilizada em ambientes de produção para realizar tarefas críticas, como envio de e-mails em massa, processamento de dados em lotes e outras operações que exigem alta performance e confiabilidade. No entanto, apesar de sua robustez e flexibilidade, não possui ferramentas nativas para testes de resiliência ou para simular falhas em tempo real, o que dificulta a avaliação do seu comportamento quando exposto a condições adversas.

1.2 OBJETIVO

O objetivo deste trabalho é investigar a resiliência e o desempenho do Celery quando exposto a falhas controladas. Para alcançar esse objetivo, serão realizados experimentos que simulam condições adversas, como falhas de rede, exaustão de memória e sobrecarga de CPU, com o intuito de observar o impacto dessas falhas na execução das tarefas e no comportamento da ferramenta. O projeto também visa monitorar e analisar métricas essenciais, como o tempo de execução das tarefas, taxa de processamento, taxa de sucesso, além de parâmetros relacionados à eficiência do sistema, como o número de tarefas em execução e pré-carregadas no worker.

Através da coleta dessas métricas, será possível avaliar o desempenho do Celery sob diferentes cenários de falha, identificar gargalos no processamento das tarefas e testar a capacidade de recuperação do sistema em situações de sobrecarga ou falha. Também faz parte do propó-

sito do estudo fornecer uma análise detalhada de como o Celery se comporta em um ambiente de produção simulado, oferecendo insights sobre sua confiabilidade e eficiência em contextos de alta demanda e falhas inesperadas.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 TESTES DE RESILIÊNCIA

São testes que fazem parte de uma abordagem que visa avaliar a resiliência de sistemas introduzindo falhas intencionais em ambientes monitorados, avaliando as consequências disso no sistema em questão. Essa prática, também conhecida como Chaos Testing, surgiu da necessidade de entender como os sistemas reagem a falhas inesperadas e garantir que eles possam continuar operando de maneira confiável mesmo sob condições adversas e inesperadas. O princípio base dessa área é realizar experimentos em ambientes controlados, podendo ser até em ambientes de produção e pré-produção, e identificar pontos fracos, gargalos e efeitos cascatas indesejados para melhorar a robustez do sistema. (GREMLIN, 2025)

Alguns ganhos notáveis do uso dessa prática são:

- Identificar falhas antes que ocorram em produção.
- Melhorar a confiança na resiliência do sistema.
- Ajudar equipes a desenvolverem estratégias eficazes de recuperação do sistema.

A popularidade dessa prática veio à tona com a Netflix, que criou o Chaos Monkey para simular falhas em sua infraestrutura na nuvem, e a partir desse experimento muito original, o conceito evoluiu para um conjunto mais amplo de práticas que são conhecidas como Chaos Engineering. (BASIRI et al., 2016)

2.2 CELERY

Celery é um sistema open-source de fila de tarefas assíncronas que permite distribuir a execução de tarefas em múltiplos workers. Ele é amplamente utilizado para processar tarefas em background das mais diversas aplicações, melhorando a escalabilidade de sistemas e lidando com workloads intensivos de processamento de dados, por exemplo.

O funcionamento do Celery se baseia em três componentes principais:

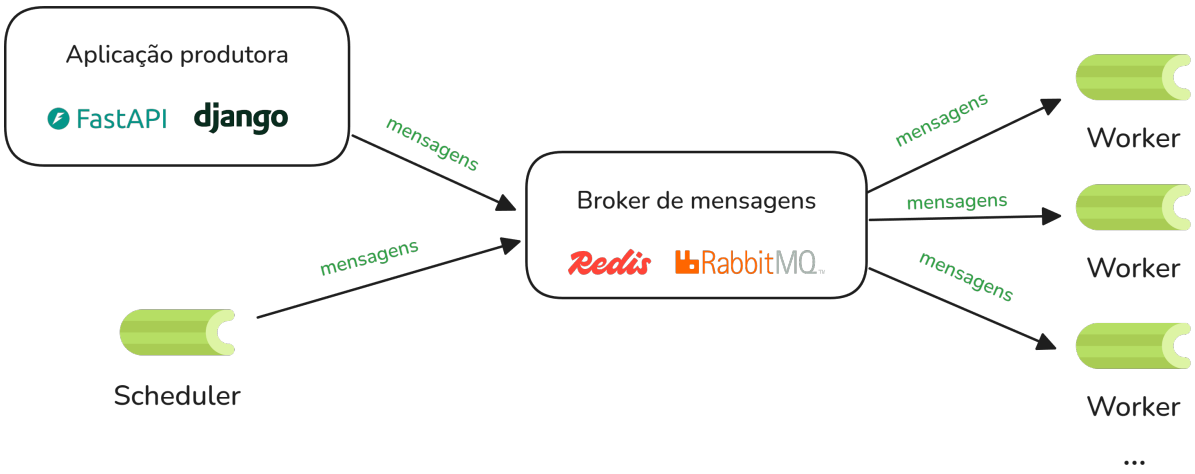

Figura 1 – Funcionamento de um sistema com Celery

Fonte: Elaborado pelo autor (2025)

- Aplicação produtora: Cria e envia as tarefas para a fila. Geralmente, é uma aplicação backend, como uma API, que utiliza o Celery para executar tarefas em segundo plano, sem bloquear o fluxo principal.
- Broker: O sistema de mensagens que armazena as tarefas criadas pela aplicação produtora ou pelo scheduler de tarefas, e as encaminha para os workers. Os mais usados são o Redis (Redis Project, 2025) e RabbitMQ (Broadcom, 2025).
- Workers: São os processos que consomem as tarefas armazenadas no broker e executam as funções definidas pela aplicação produtora.

É um sistema idealizado para lidar com grandes volumes de tarefas assíncronas de maneira eficiente, distribuindo a carga de processamento entre vários workers. Também suporta dois tipos de processamento de tarefas: tempo real e agendamento.

2.3 COMO FALHAS PODEM IMPACTAR O CELERY

Sistemas de processamento assíncronos estão sujeitos a diferentes tipos de falhas inesperadas que podem impactar o comportamento e resultados da aplicação, por isso é importante testar a resiliência desse tipo de sistema diante de situações inesperadas do mundo real.

Alguns pontos críticos de impacto que um sistema com Celery pode sofrer são:

1. Exaustão de recursos

Como qualquer sistema, ele depende dos recursos disponíveis em seu ambiente para operar de forma esperada, e em casos de limites ultrapassados — como uso excessivo de CPU, memória, disco ou tempo de execução — é comum ocorrer falhas ou interrupções no funcionamento dos workers e, consequentemente, no processamento de tarefas.

2. Perda de mensagens

   Sob condições adversas, falhas de workers podem resultar na perda de mensagens, o que é perigoso para aplicações críticas.

3. Observabilidade

   Por processar as tarefas de forma assíncrona, problemas de latência, perda de mensagens ou falhas intermitentes podem passar despercebidos sem uma observabilidade adequada, comprometendo a capacidade de diagnosticar e resolver problemas rapidamente.

3 METODOLOGIA

3.1 ARQUITETURA DO EXPERIMENTO

A arquitetura do experimento foi desenhada para simular e testar o comportamento de um sistema Celery experimental sob diferentes condições de carga e falhas. Para isso, é feito o uso do Docker Compose, que orquestra múltiplos containers Docker, onde cada serviço tem sua responsabilidade no experimento:

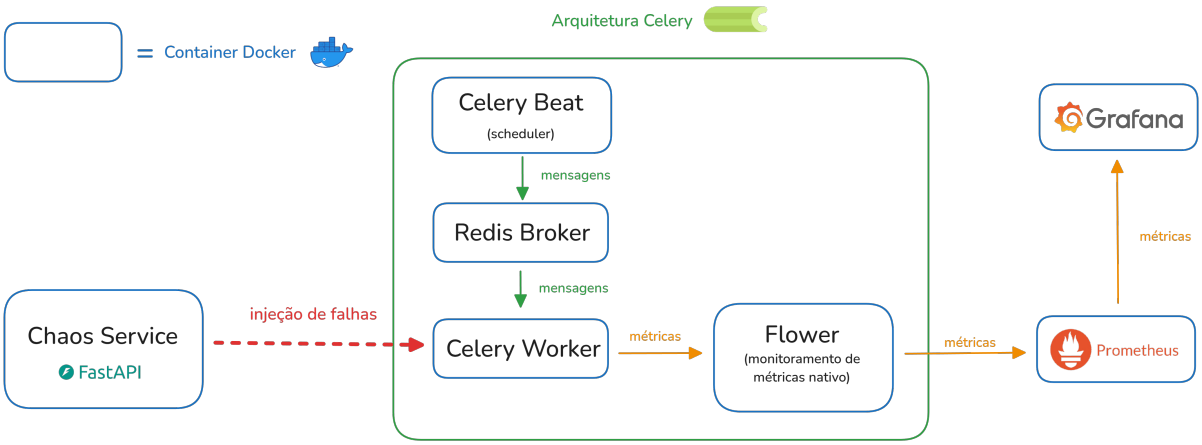

Figura 2 – Fluxograma da arquitetura do experimento

Fonte: Elaborado pelo autor (2025)

1. Celery Worker

   É quem está processando as tarefas experimentais projetadas para simular diferentes tipos de uso de recursos em um sistema do mundo real.

2. Celery Beat

   Responsável pelo disparo das tarefas agendadas para o worker de forma periódica.

3. Redis (Redis Project, 2025)

   O broker de mensagens configurado para o Celery.

4. Chaos Service

   É uma API desenvolvida com FastAPI (RAMíREZ, 2025) que contém endpoints específicos de injeção de falhas no sistema Celery e seus componentes.

5. Flower (MOVSISYAN, 2023)

   Ferramenta de monitoramento em tempo real do Celery, disponibilizando métricas como tempo de execução, taxa de sucesso e falha, tempo médio de resposta, e estado dos workers.

6. Prometheus (Prometheus Authors, 2025)

   Ferramenta que coleta e armazena métricas de desempenho do sistema, extraindo dados do serviço Flower.

7. Grafana (Grafana Labs, 2025)

   Utilizando os dados coletados pelo Prometheus, o Grafana é uma ferramenta usada para exibir um dashboard de observabilidade, proporcionando uma visão detalhada do comportamento do sistema.

A cada minuto, o serviço Celery Beat dispara uma tarefa pai executada pelo Celery Worker, que por sua vez adiciona todas as tarefas de simulação da carga realista do sistema na mesma fila de execução do worker utilizado.

3.2 TAREFAS CRIADAS

As tarefas que estão sendo executadas no worker do Celery foram projetadas para focarem no uso de diferentes recursos do sistema, simulando casos de uso do mundo real. Isso nos ajudará a entender como falhas dedicadas ao desgaste neste tipo de recurso afetará a realização das tarefas.

1. Intensiva de Rede

   Efetua uma requisição HTTP a uma API externa e retorna os dados extraídos. É semelhante a carga de recursos de uma consulta a uma API de pagamento para verificar a aprovação de transações em tempo real.

2. Intensiva de Memória

   Criação de uma matriz de números aleatórios de grande porte e são realizados cálculos pesados sobre essa matriz, simulando uma carga como o processamento de grandes volumes de dados.

3. Intensiva de CPU

   Realiza cálculos pesados para encontrar números primos até um limite definido, consumindo a CPU disponível para o serviço, simulando uma carga como a de efetuar cálculos complexos em simulações científicas.

3.3 FALHAS INJETADAS

As falhas introduzidas serão essenciais nesta avaliação experimental para avaliar o impacto de diferentes tipos de degradação de recursos no desempenho e na resiliência do sistema Celery utilizado. Vamos poder observar como o sistema lida com a recuperação e a continuidade da execução das tarefas durante a injeção destas falhas:

1. Delay de rede no Redis

   É criado um atraso na comunicação entre o broker Redis e o worker do Celery. A falha é implementada utilizando o comando *tc qdisc add*, que adiciona um delay de rede ao Redis.

2. Exaustão de Memória

   Tornamos escassos os recursos de memória livres para execução de tarefas em um worker através da ferramenta stress (WATERLAND, 2008), que aloca constantemente memória, consumindo grande parte dos recursos disponíveis.

3. Exaustão de CPU

   Semelhante à falha de exaustão de memória, é executada a ferramenta stress dentro do container do worker, que irá realizar cálculos pesados e aumentar significativamente o uso da CPU disponível no serviço.

3.4 MÉTRICAS COLETADAS

A coleta dessas métricas é essencial para analisar o desempenho e a resiliência do sistema porque irá nos permitir identificar gargalos e falhas no processamento de tarefas.

1. Distribuição do Tempo de Execução da Tarefa

Mede o tempo de execução de cada tarefa, distribuindo-o para entender a variação e o comportamento da carga de trabalho de um worker.

2. Taxa de Processamento de Tarefas

   É o número de tarefas processadas por unidade de tempo, o que irá nos ajudar a avaliar a capacidade do sistema em processar tarefas.

3. Tempo Médio de Execução da Tarefa no Worker

   Calcula o tempo médio de execução das tarefas.

4. Tempo de Pré-carregamento da Tarefa no Worker

   Mede o tempo entre o momento em que uma tarefa é atribuída ao worker e o momento em que o worker começa a executá-la. Isso é importante para entender a latência no processamento das tarefas e detectar possíveis gargalos na fila.

5. Taxa de Sucesso das Tarefas

   A proporção de tarefas concluídas com sucesso em relação ao total de tarefas executadas.

Entre todas as métricas coletadas, a taxa de sucesso será considerada a métrica mais relevante para a análise de resiliência nesse estudo, pois reflete diretamente a capacidade do sistema de manter sua funcionalidade sob condições adversas. As demais métricas complementam essa análise ao indicar se o ambiente de execução realmente enfrentou situações adversas e como o sistema respondeu a elas.

4 RESULTADOS E DISCUSSÃO

4.1 EXPERIMENTO BASELINE

A cada 1 minuto, são geradas 20 tarefas de requisição HTTP, 2 tarefas intensivas de CPU e 2 tarefas intensivas de memória. Todas as tarefas são processadas por um único worker, que possui 1GB de memória disponível e possui nível 8 de concorrência de tarefas (baseado no número de núcleos da máquina usada). Essa configuração inicial serve como referência para entender o comportamento do sistema em termos de tempo de execução, taxa de processamento e uso de recursos, antes da introdução de falhas, e assim, temos uma visão clara de como o ambiente experimental configurado opera sem interferências externas, e o que ocorre após a injeção de um stress controlado.

Os resultados baseline foram os seguintes:

- **Taxa de Sucesso das Tarefas:** 100%

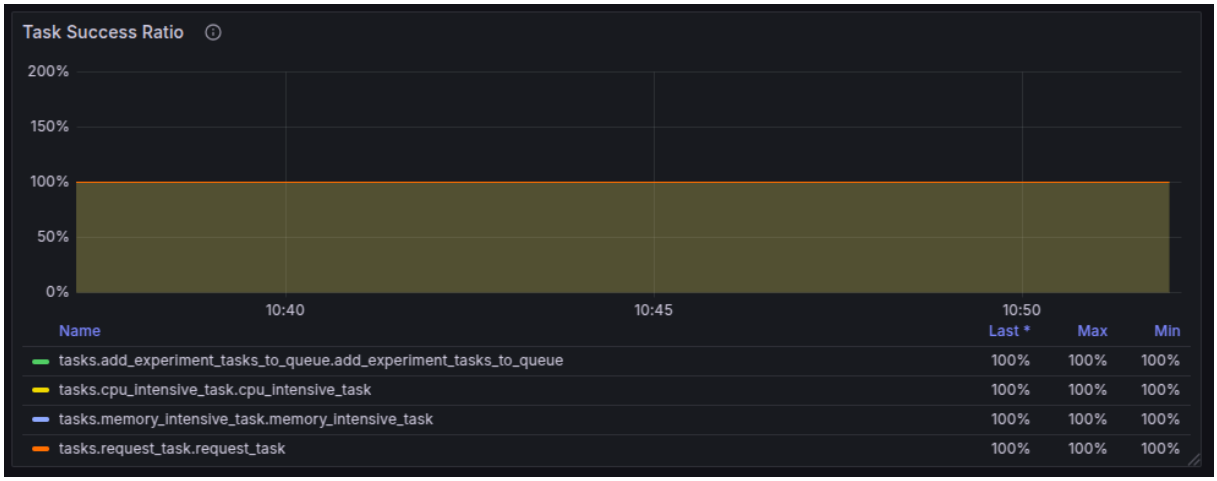

Figura 3 – Experimento Baseline - Gráfico da taxa de sucesso das tarefas

Fonte: Elaborado pelo autor (2025)

- **Distribuição do Tempo de Execução:** Bastante estável durante o experimento, com a tarefa de uso intensivo de memória se destacando como a mais demorada, seguido da tarefa de uso intensivo de CPU. Já a tarefa de requisição HTTP possui uma média de distribuição perto de 0.

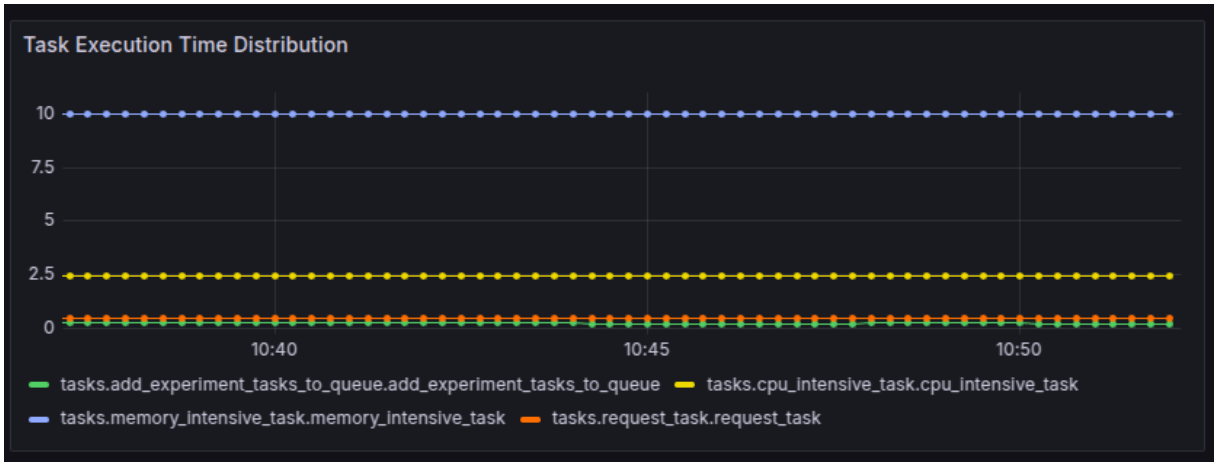

Figura 4 – Experimento Baseline - Gráfico da distribuição do tempo de execução

Fonte: Elaborado pelo autor (2025)

- **Taxa de Processamento de Tarefas:** Resultado bem constante para o experimento, devemos lembrar que como são criadas mais tasks de requisições HTTP do que as outras, é natural que ela seja a com mais taxa de processamento por unidade de tempo, enquanto as outras ficam abaixo. As oscilações são explicadas pela frequência de criação de tasks, que é a cada 1 minuto.

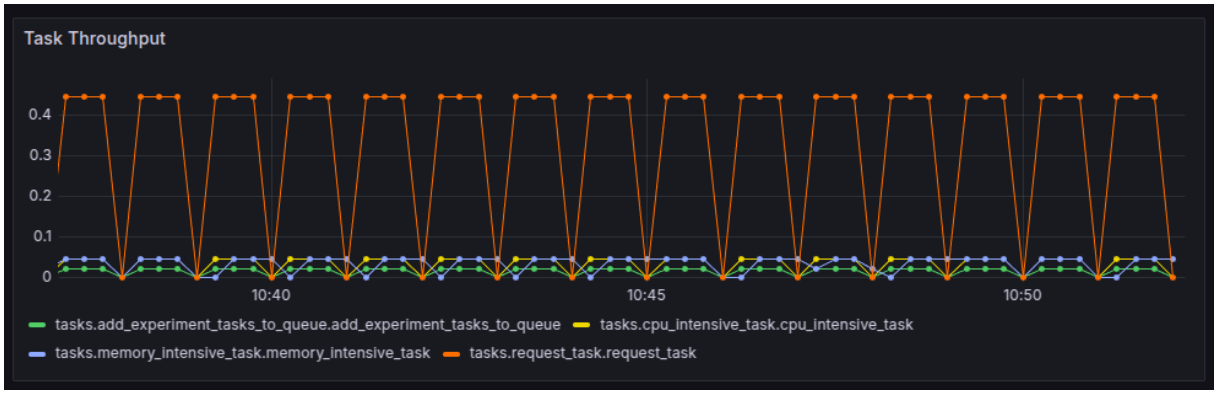

Figura 5 – Experimento Baseline - Gráfico da taxa de processamento das tarefas

Fonte: Elaborado pelo autor (2025)

- **Tempo Médio de Execução da Tarefa no Worker:** Semelhante ao resultado de tempo de execução, temos um gráfico relativamente constante demonstrando que as tarefas de requisição HTTP têm um tempo de execução muito rápido, enquanto as tarefas intensivas de CPU e memória têm tempos de execução mais altos, principalmente a tarefa de memória, como esperado devido à maior complexidade computacional exigida nela.

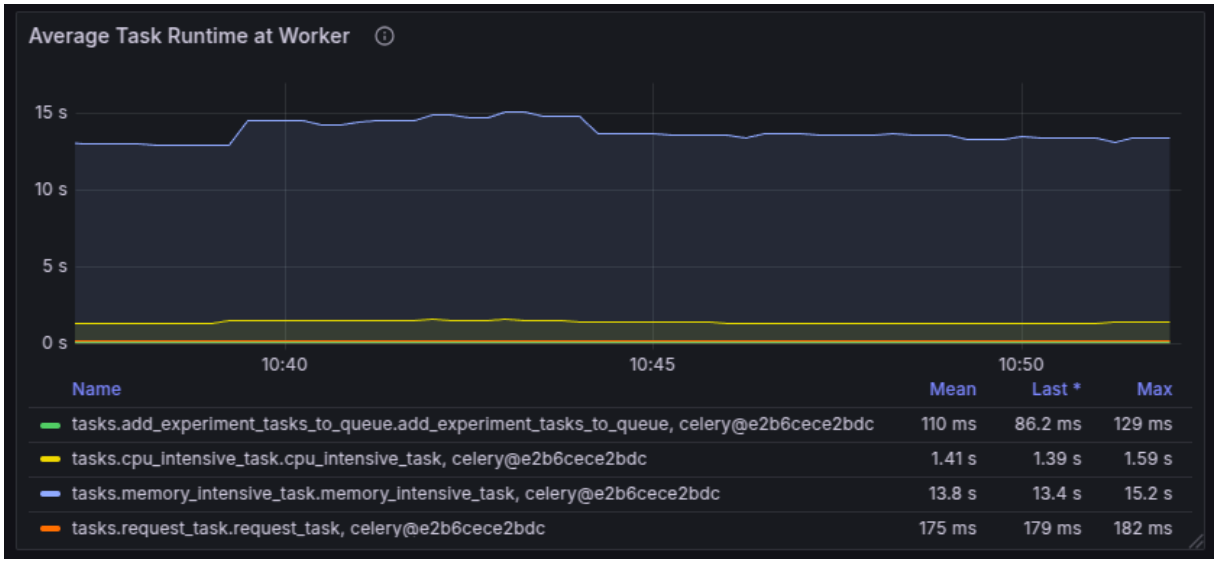

Figura 6 – Experimento Baseline - Gráfico do tempo médio de execução das tarefas

Fonte: Elaborado pelo autor (2025)

- **Tempo de Pré-carregamento da Tarefa no Worker:** O pré-carregamento das tarefas está eficiente, com tempos muito baixos (próximos de 0ms) para as tarefas de requisição HTTP e CPU, enquanto a tarefa de memória apresenta uma variação mais alta devido a seu tempo elevado de execução, mas ainda dentro de uma faixa razoável.

Figura 7 – Experimento Baseline - Gráfico do tempo de pré-carregamento das tarefas

Fonte: Elaborado pelo autor (2025)

4.2 IMPACTOS DAS FALHAS NAS MÉTRICAS

As falhas são injetadas individualmente após 15 minutos do início da execução da carga baseline, a fim de isolar seus efeitos no sistema. A seguir, são analisadas as consequências específicas de cada falha.

4.2.1 Delay de rede

- **Taxa de Sucesso das Tarefas:** Se manteve em 100%.

Figura 8 – Experimento com Delay de Rede - Gráfico da taxa de sucesso das tarefas

Fonte: Elaborado pelo autor (2025)

- **Distribuição do Tempo de Execução:** É possível identificar um aumento repentino no tempo de execução das tarefas, principalmente na tarefa pai que adiciona as tarefas de simulação de carga na fila, que subiu de próximo a 0 para 10 segundos. Essa tarefa, por ser iniciada programaticamente pelo Celery Beat, necessita de uma latência boa entre as partes para um tempo de processamento curto. As tarefas de requisição HTTP e de uso da CPU também apresentam um aumento no tempo de execução, mas de forma menos aguda.

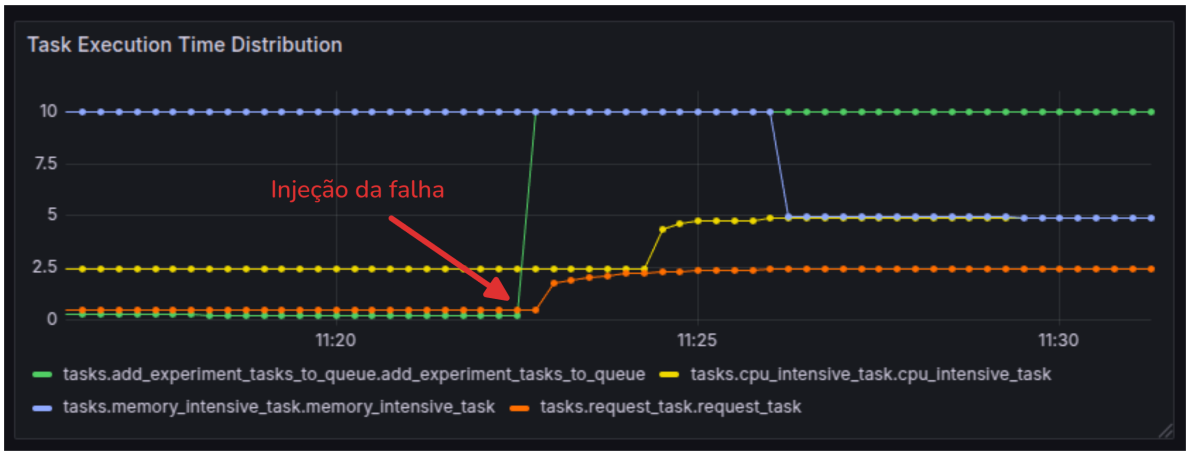

Figura 9 – Experimento com Delay de Rede - Gráfico da distribuição do tempo de execução

Fonte: Elaborado pelo autor (2025)

- **Taxa de Processamento de Tarefas:** Observamos uma diminuição significativa no throughput de todas as tarefas, especialmente na tarefa de requisição HTTP, e criação de um comportamento menos uniforme.

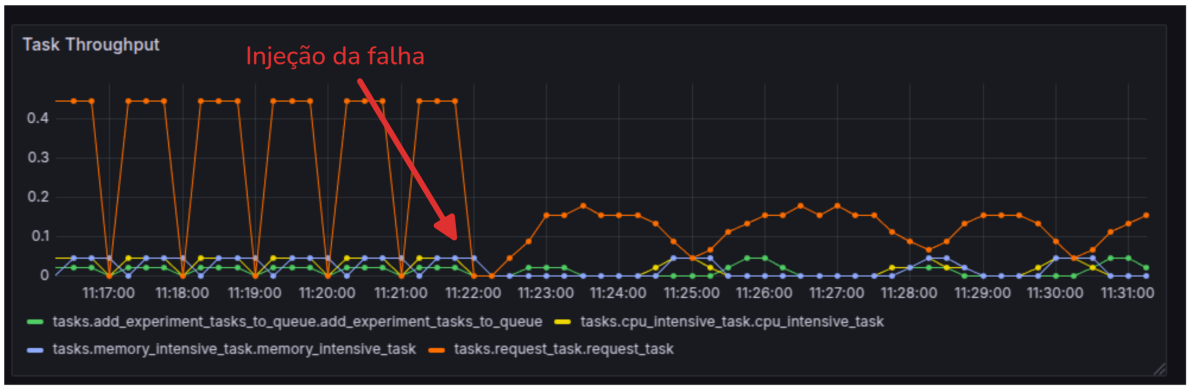

Figura 10 – Experimento com Delay de Rede - Gráfico da taxa de processamento das tarefas

Fonte: Elaborado pelo autor (2025)

- **Tempo Médio de Execução da Tarefa no Worker:** Aumento abrupto no tempo de execução das tarefas, especialmente na tarefa pai de criação das tarefas que simulam a carga no ambiente.

Figura 11 – Experimento com Delay de Rede - Gráfico do tempo médio de execução das tarefas

Fonte: Elaborado pelo autor (2025)

- **Tempo de Pré-carregamento da Tarefa no Worker:** Aumento significativo no tempo de pré-carregamento, impactando o tempo de espera antes da execução. A injeção da falha tornou a sincronização e o gerenciamento das tarefas menos eficientes.

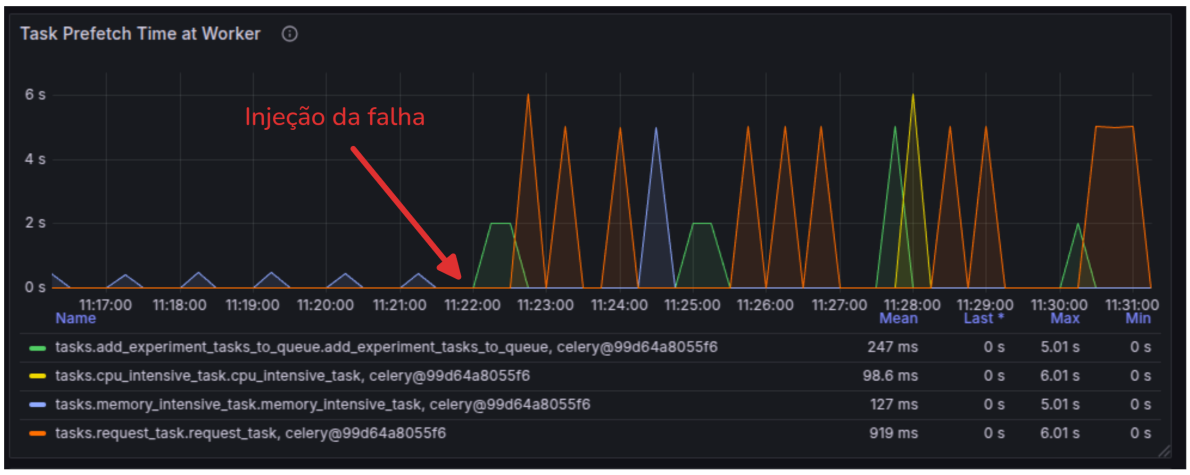

Figura 12 – Experimento com Delay de Rede - Gráfico do tempo de pré-carregamento das tarefas

Fonte: Elaborado pelo autor (2025)

4.2.2 Exaustão de CPU

- **Taxa de Sucesso das Tarefas:** Se manteve em 100%.

Figura 13 – Experimento com Exaustão de CPU - Gráfico da taxa de sucesso das tarefas

Fonte: Elaborado pelo autor (2025)

- **Distribuição do Tempo de Execução:** Aumento instantâneo na curva de distribuição de tempo apenas da tarefa de consumo de CPU, que se estabilizou conforme o tempo em um patamar um pouco mais elevado.

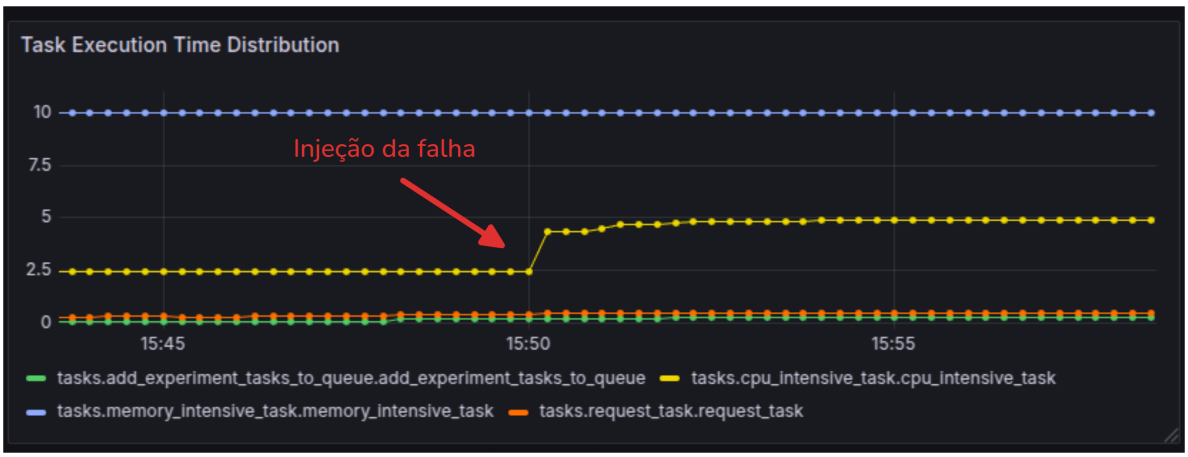

Figura 14 – Experimento com Exaustão de CPU - Gráfico da distribuição do tempo de execução

Fonte: Elaborado pelo autor (2025)

- **Taxa de Processamento de Tarefas:** Sem mudanças perceptíveis.

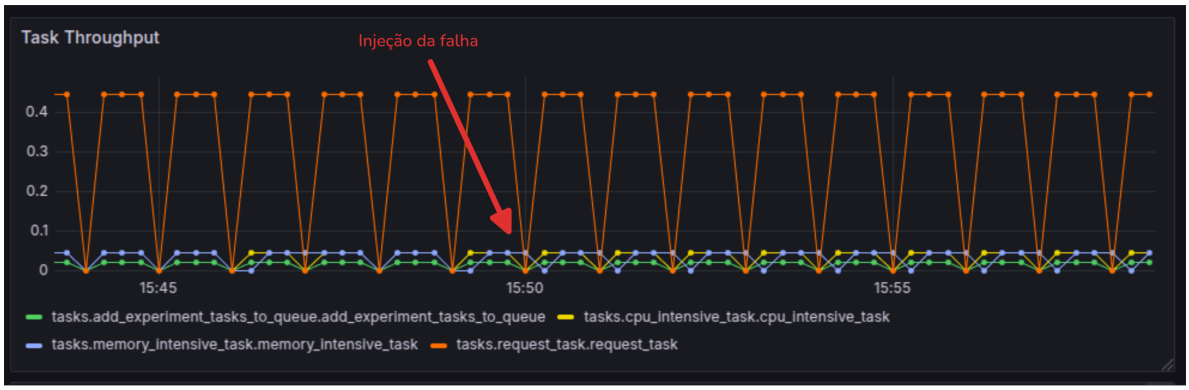

Figura 15 – Experimento com Exaustão de CPU - Gráfico da taxa de processamento das tarefas

Fonte: Elaborado pelo autor (2025)

- **Tempo Médio de Execução da Tarefa no Worker:** Aumento no tempo de execução das tarefas com consumo de CPU e de memória, mas que se estabilizam com o tempo.

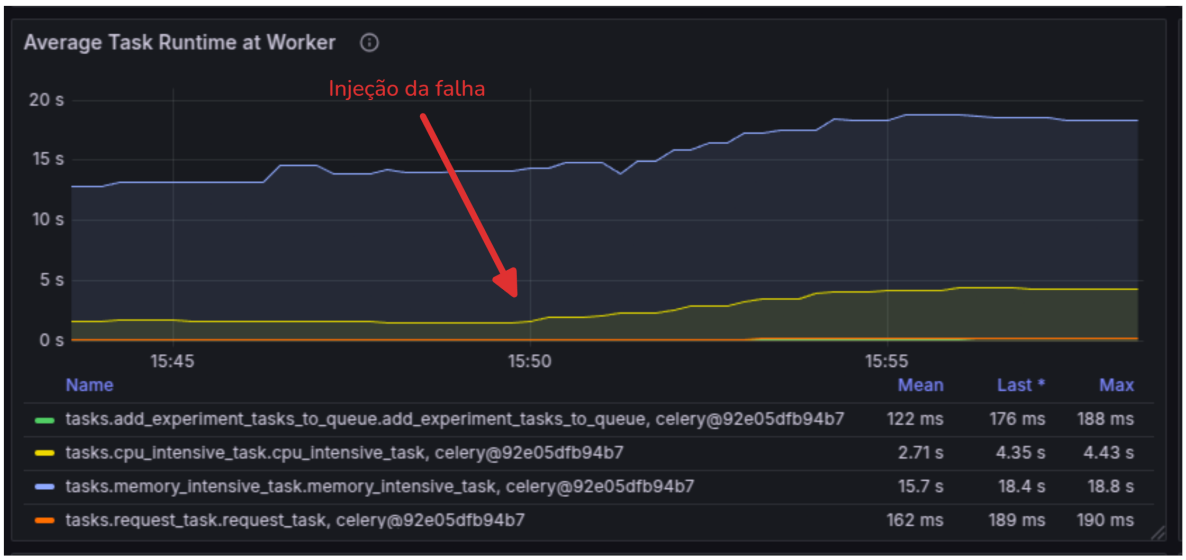

Figura 16 – Experimento com Exaustão de CPU - Gráfico do tempo médio de execução das tarefas

Fonte: Elaborado pelo autor (2025)

- **Tempo de Pré-carregamento da Tarefa no Worker:** Tarefa intensiva de memória apresentou mais picos de pré-carregamento.

Figura 17 – Experimento com Exaustão de CPU - Gráfico do tempo de pré-carregamento das tarefas

Fonte: Elaborado pelo autor (2025)

4.2.3 Exaustão de memória

4.2.3.1 Configuração padrão

- **Taxa de Sucesso das Tarefas:** Tarefa de consumo de memória tem uma queda imediata na taxa de sucesso devido a vários erros abruptos de encerramento do Worker, causado pelo uso de memória excedendo os recursos disponíveis no container.

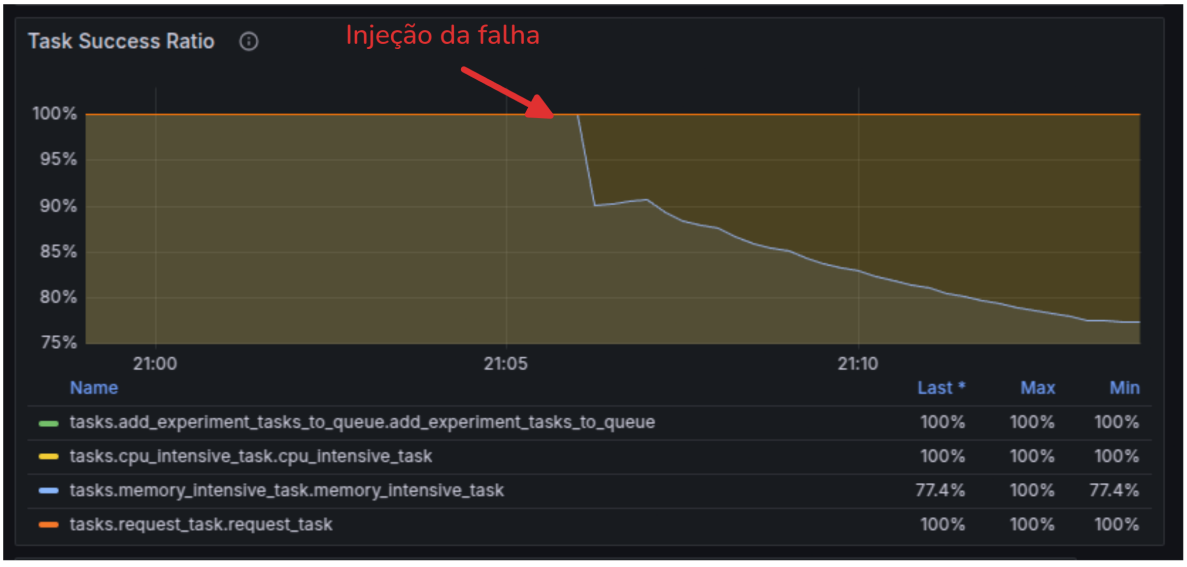

Figura 18 – Experimento com Exaustão de Memória - Gráfico da taxa de sucesso das tarefas

Fonte: Elaborado pelo autor (2025)

- **Distribuição do Tempo de Execução:** Queda não imediata na distribuição de tempo na tarefa de memória devido a diminuição da taxa de sucesso, e um aumento, também não imediato, na distribuição do tempo da tarefa de CPU.

Figura 19 – Experimento com Exaustão de Memória - Gráfico da distribuição do tempo de execução

Fonte: Elaborado pelo autor (2025)

- **Taxa de Processamento de Tarefas:** Taxa da tarefa intensiva de memória tem queda de perfomance imediata.

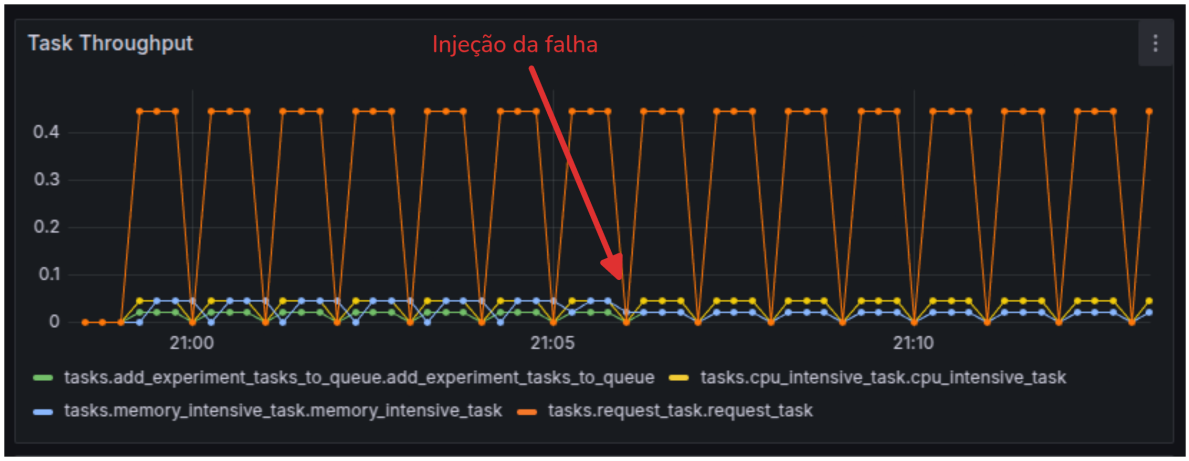

Figura 20 – Experimento com Exaustão de Memória - Gráfico da taxa de processamento das tarefas

Fonte: Elaborado pelo autor (2025)

- **Tempo Médio de Execução da Tarefa no Worker:** Queda no tempo médio de

execução da tarefa de memória devido a queda da taxa de sucesso, e aumento leve no de CPU.

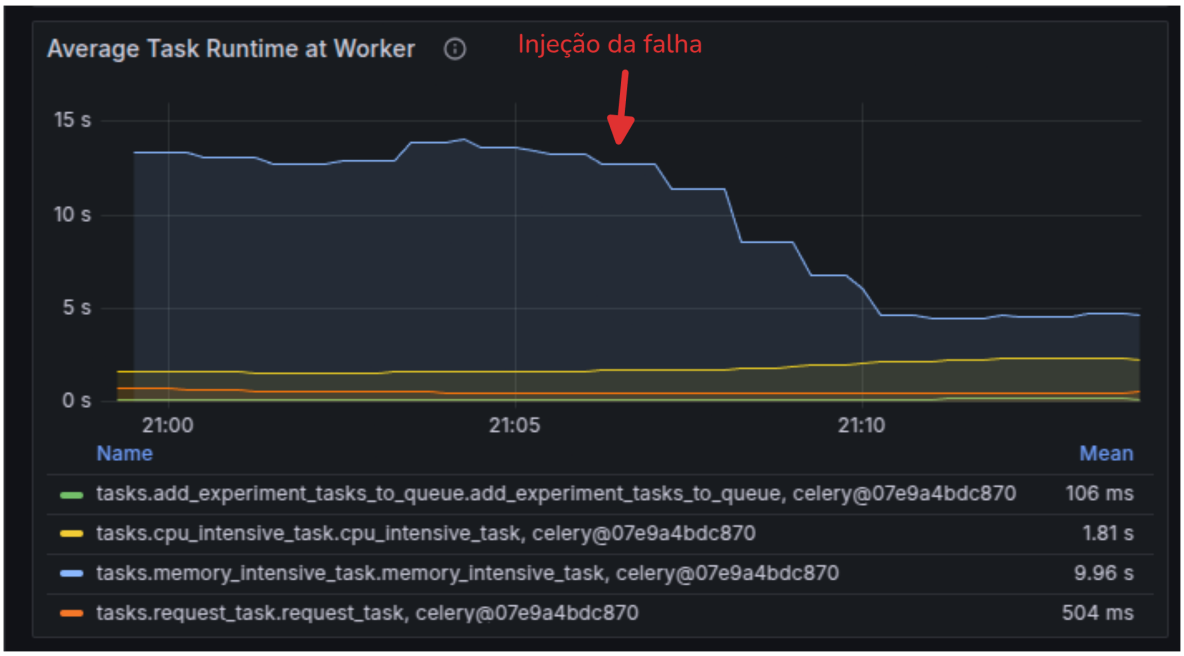

Figura 21 – Experimento com Exaustão de Memória - Gráfico do tempo médio de execução das tarefas

Fonte: Elaborado pelo autor (2025)

- **Tempo de Pré-carregamento da Tarefa no Worker:** Tempo de pré-carregamento das tarefas de memória é zerado devido aos erros de terminação de worker.

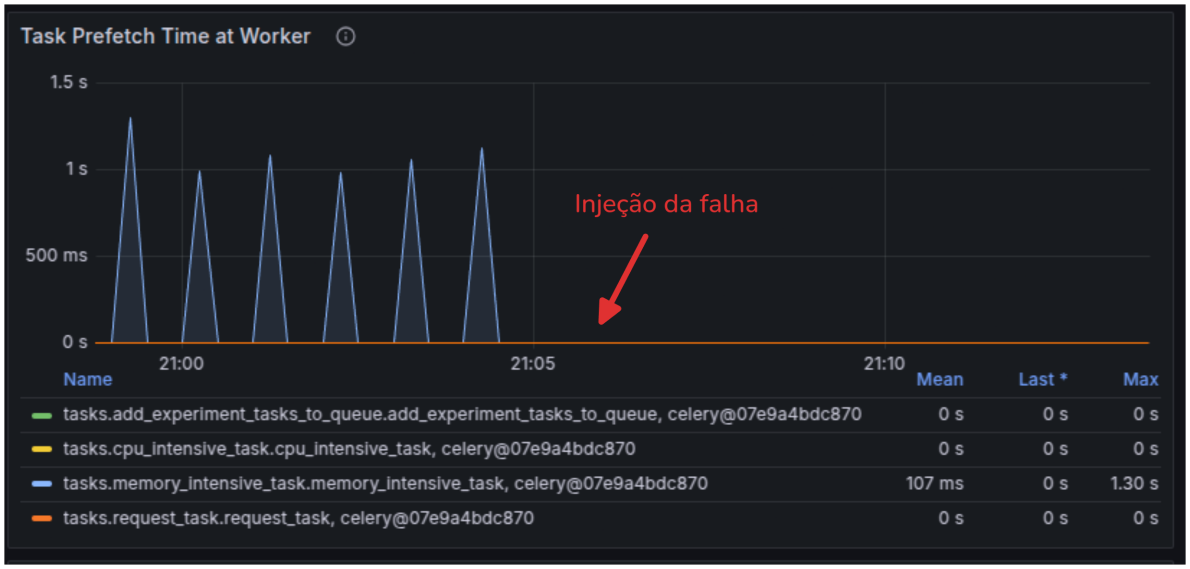

Figura 22 – Experimento com Exaustão de Memória - Gráfico do tempo de pré-carregamento das tarefas

Fonte: Elaborado pelo autor (2025)

4.2.3.2 Acks Late

Nos resultados apresentados após a injeção da falha de exaustão de memória, foi possível observar que os impactos estavam diretamente relacionados aos erros de terminação de worker devido ao uso excessivo de memória (Out of Memory), e que as tarefas que estavam sendo executadas durante esses momentos foram perdidas. No entanto, a documentação oficial do Celery (Celery Project, 2023) recomenda ajustes de configuração específicos para evitar essa perda de tarefas, através da ativação das propriedades:

- `task_acks_late`
- `task_reject_on_worker_lost`

Com essas configurações ativadas, o worker só sinaliza que uma tarefa foi concluída após sua finalização, então, em caso de terminação abrupta, a tarefa segue marcada como pendente e não é perdida pelo processamento, sendo reinserida na fila. Após essa modificação, todas as métricas passaram a não ser mais afetadas pela falha de exaustão de memória, demonstrando um excelente novo resultado de avaliação da resiliência do Celery em relação a falha de memória.

4.3 ANÁLISE DE IMPACTO

Com a observabilidade criada no experimento através do Grafana, vemos que o sistema demonstrou boa resiliência quando configurado corretamente. No caso das falhas de exaustão de CPU e adição de delay de comunicação, os impactos observados foram principalmente no aumento do tempo de execução das tarefas, com uma redução no throughput e aumento do tempo de pré-carregamento, mas foram estabilizados ao longo do tempo e não trouxeram falhas críticas no sistema. A tarefa pai, responsável por adicionar as tarefas de simulação de carga à fila, foi a mais impactada, apresentando um aumento substancial no tempo de execução, indicando que a latência afetou diretamente o gerenciamento de tarefas. Apesar disso, o sistema conseguiu se manter estável, mesmo com pioras em várias métricas relacionadas à performance.

Quando configuradas as propriedades `task_acks_late` e `task_reject_on_worker_lost` para inibir a perda de tarefas durante falhas, os resultados mostraram uma significativa melhora

de resiliência e estabilidade relacionados a exaustão de memória, que havia sido a falha injetada mais impactante originalmente. As tarefas que anteriormente foram perdidas devido a erros abruptos de encerramento do Worker pelos limites de memória, agora são reprocessadas sem comprometer o sistema, permitindo que o Celery se recupere sem perdas de dados e sem pioras perceptíveis no desempenho.

5 CONCLUSÃO

Este trabalho teve como objetivo investigar a resiliência do Celery quando exposto a falhas controladas, especificamente falhas de rede, exaustão de memória e sobrecarga de CPU. Foi possível observar o impacto dessas falhas no comportamento do Celery por meio de diferentes experimentos e uso da observabilidade para acompanhamento dos resultados, através de métricas como tempo de execução das tarefas, taxa de processamento e taxa de sucesso.

A análise dos resultados mostrou que o Celery, sem configurações específicas, demonstrou boa resiliência, especialmente ao lidar com falhas de exaustão de CPU e latência de comunicação, que afetaram o tempo de execução das tarefas e o throughput sem comprometer o sucesso das tarefas, se mantendo estável ao longo do tempo. A implementação das configurações `task_acks_late` e `task_reject_on_worker_lost` trouxeram uma melhora significativa na resiliência diante de cenários com exaustão de memória, que era o caso mais impactante negativamente. Essas configurações permitiram que o Celery evitasse a perda de tarefas durante falhas, garantindo a continuidade do processamento e uma recuperação eficaz sem perda de dados.

Em suma, este estudo demonstrou que o Celery pode ser uma ferramenta robusta e resiliente em ambientes de alta demanda e falhas controladas, com as configurações apropriadas para mitigar os efeitos de falhas inesperadas e manter a eficiência do sistema.

6 TRABALHOS FUTUROS

Como grande parte das práticas adotadas de Chaos Testing em ambientes experimentais, existem limitações a partir do momento em que estamos executando simulações em um ambiente controlado e com recursos limitados, o que pode não refletir a complexidade e variabilidade de um sistema de produção real. Além disso, o número de workers foi limitado a um único worker em todos os testes, o que pode ter impactado a escalabilidade e a distribuição de carga do sistema. Futuros experimentos poderiam incluir diferentes configurações de workers e uma maior variedade de falhas para avaliar o comportamento do Celery em cenários mais complexos e diversificados.

REFERÊNCIAS

BASIRI, A.; BEHNAM, N.; ROOIJ, R. de; HOCHSTEIN, L.; KOSEWSKI, L.; REYNOLDS, J.; ROSENTHAL, C. Chaos engineering. *IEEE Software*, v. 33, n. 3, p. 35–41, 2016.

Broadcom. *RabbitMQ Documentation*. 2025. Acesso em: 27 mar. 2025. Disponível em: <https://www.rabbitmq.com/docs>.

Celery Project. *Celery Documentation*. 2023. Acesso em: 27 mar. 2025. Disponível em: <https://docs.celeryq.dev/>.

Celery Project. *Celery - Distributed Task Queue*. 2025. Acesso em: 27 mar. 2025. Disponível em: <https://github.com/celery/celery>.

Grafana Labs. *Grafana Documentation*. 2025. Acesso em: 27 mar. 2025. Disponível em: <https://grafana.com/docs/>.

GREMLIN. *What is Chaos Engineering?* 2025. Acesso em: 27 mar. 2025. Disponível em: <https://www.gremlin.com/chaos-engineering>.

MOVSISYAN, M. *Flower: Celery monitoring tool*. 2023. Acesso em: 27 mar. 2025. Disponível em: <https://flower.readthedocs.io/>.

Prometheus Authors. *Prometheus Documentation*. 2025. Acesso em: 27 mar. 2025. Disponível em: <https://prometheus.io/docs/>.

RAMíREZ, S. *FastAPI Documentation*. 2025. Acesso em: 27 mar. 2025. Disponível em: <https://fastapi.tiangolo.com/>.

Redis Project. *Redis Documentation*. 2025. Acesso em: 27 mar. 2025. Disponível em: <https://redis.io/docs/>.

THALLAPALLY, N. Enhancing distributed systems with message queues: Architecture, benefits, and best practices. *Journal of Electrical Systems*, 3 2025. Disponível em: <https://journal.esrgroups.org/jes/article/view/8333>.

WATERLAND, A. *stress - a workload generator for POSIX systems*. 2008. Disponível em: <https://github.com/resurrecting-open-source-projects/stress>.