

Actor Model Performance in Different Scenarios

Programação Avançada
Grupo 5
David Carreira
João Ganhão



Introdução

- Objetivo: Comparar desempenho entre três modelos concorrentes:
- Contexto e Importância: Esta análise visa avaliar a eficiência de diferentes abordagens de concorrência, fundamental para o desenvolvimento de sistemas escaláveis e responsivos
- Thread por tarefa:
Processamento sequencial direto.
- ThreadPool:
Reutilização eficiente de threads.
- Modelo de Atores:
Concorrência baseada em mensagens.
- Métricas de Desempenho Específicas:
 - Tempo de Execução (Latência e Throughput)
 - Consumo de Memória (Uso de Heap e Stack)
 - Escalabilidade (Fator de Aumento de Carga)

Modelos Concorrentes

- Thread por tarefa:
- Uma thread dedicada é criada para cada nova tarefa. As threads compartilham recursos do processo, mas possuem suas próprias pilhas e registradores. Simples, mas com alto overhead de criação/destruição.
- Vantagens: Simples de implementar para poucas tarefas.
- Desvantagens: Alto custo de recursos problemas de escalabilidade risco de exaustão de threads.

Modelos Concorrentes

- ThreadPool Reutilização de um conjunto fixo de threads para processar tarefas de uma fila. Evita o overhead de criação frequente de threads.
- Vantagens: Melhor gestão de recursos menor overhead desempenho melhorado para muitas tarefas curtas.
- Desvantagens: Risco de deadlock se threads bloquearem ajuste do tamanho do pool pode ser complexo fila pode ser gargalo.

Modelos Concorrentes

- Modelo de Atores Unidades independentes (Atores) comunicam-se de forma assíncrona através de mensagens imutáveis. Cada ator possui seu próprio estado e processa mensagens sequencialmente de sua caixa de correio
- Vantagens: Sem estado compartilhado (evita locks) modelo natural para sistemas distribuídos tolerância a falhas.
- Desvantagens: Overhead de troca de mensagens pode ser complexo de codificar requer mudança de paradigma.

Metodologia



Parâmetros do Teste



Número de Tarefas: Escala de 100 a 10.000



Repetições e Medições: Múltiplas execuções, medições em intervalos regulares



Ambiente Padronizado: Mesmo hardware e ambiente Python 3.x



Tipos de Tarefas e Exemplos



Cálculo Simples: Operações matemáticas básicas (e.g., adição, multiplicação)



CPU Intensivo: Cálculo de números primos, processamento de matrizes



I/O: Leitura/escrita de arquivos, requisições de rede simuladas



Ambiente de Teste



Especificações de Hardware

CPU: Intel Core i5-10210U (4 Cores, 8 Threads).

Adequado para avaliar concorrência com múltiplas threads.

- RAM: 8 GB DDR4.

Suficiente para as cargas de trabalho propostas, evitando gargalos de memória.

- SO: Windows 10/11 64 bits.

Ambiente de desktop comum e representativo.



Ambiente de Teste



Python: Versão 3.13.9.



Consideração Crítica: O Global Interpreter Lock (GIL) limita a execução paralela real de threads em tarefas intensivas de CPU.



Bibliotecas: 'psutil' (monitoramento de recursos), 'pykka' (Modelo de Atores).



Impacto do Ambiente: O número de núcleos



e o GIL influenciarão diretamente o desempenho, especialmente em tarefas CPU-bound.



Resultados Tempo



Modelo de Atores manteve tempos baixos mesmo com 10.000 tarefas, demonstrando a melhor eficiência

Modelo de Atores: melhor desempenho e escalabilidade geral, ideal para cargas de trabalho variáveis.

ThreadPool: bom para tarefas leves, com limitações em alta escala.

Thread por tarefa: pior escalabilidade, tempos de execução significativamente maiores.



Resultados - Memória



Memory Usage (Peak) & Efficiency Metrics

Actor Model: 45MB peak, High Efficiency, Linear Growth Pattern

ThreadPool: 78MB peak, Moderate Efficiency

Thread per task: 156MB peak, Low Efficiency, Exponential Growth Pattern.

Why Thread Creation Impacts Memory

Significant overhead per thread

Stack space allocation for each thread

Context switching costs
Efficiency Winner: Actor Model

Criação massiva de threads impacta recursos do sistema e estabilidade

Tabela 1 - Calculo simples

| Nº ficheiros | <i>Modelo de Atores</i> | <i>Thread por tarefa</i> | <i>ThreadPool</i> |
|--------------|-------------------------|--------------------------|-------------------|
| 100 | 0.01 s | 0.06 s | 0.00 s |
| 500 | 0.04 s | 0.08 s | 0.01 s |
| 1000 | 0.07 s | 0.20 s | 0.01 s |
| 5000 | 0.31 s | 0.76 s | 0.05 s |
| 10000 | 0.62 s | 1.60 s | 0.10 s |

Tabela 2 - Calculo de Ficheiros (CPU elevado)

| Nº ficheiros | <i>Modelo de Atores</i> | <i>Thread por tarefa</i> | <i>ThreadPool</i> |
|--------------|-------------------------|--------------------------|-------------------|
| 100 | 0.01 s | 1.71 s | 1.59 s |
| 500 | 0.03 s | 8.40 s | 8.50 s |
| 1000 | 0.06 s | 16.89 s | 17.01 s |
| 5000 | 0.32 s | 79.41 s | 79.51 s |
| 10000 | 0.61 s | 194.80 s | 162.83 s |

Tabela 3 - Operações de I/O

| Nº ficheiros | <i>Modelo de Atores</i> | <i>Thread por tarefa</i> | <i>ThreadPool</i> |
|--------------|-------------------------|--------------------------|-------------------|
| 100 | 0.01 s | 0.03 s | 0.21 s |
| 500 | 0.03 s | 0.11 s | 1.05 s |
| 1000 | 0.06 s | 0.17 s | 2.15 s |
| 5000 | 0.29 s | 0.70 s | 10.75 s |
| 10000 | 0.58 s | 1.49 s | 21.52 s |

B. Tabelas de Memoria (s):

Tabela 1 - Calculo simples

| Nº ficheiros | <i>Modelo de Atores</i> | <i>Thread por tarefa</i> | <i>ThreadPool</i> |
|---------------------|--------------------------------|---------------------------------|--------------------------|
| 100 | +0.15 MB | +0.20 MB | +0.08 MB |
| 500 | +0.15 MB | +1.18 MB | +0.25 MB |
| 1000 | +0.13 MB | +2.14 MB | +0.50 MB |
| 5000 | +0.21 MB | +11.43 MB | +1.64 MB |
| 10000 | +0.16 MB | +21.70 MB | +8.47 MB |

Tabela 2 - Calculo de Ficheiros (CPU elevado)

| Nº ficheiros | <i>Modelo de Atores</i> | <i>Thread por tarefa</i> | <i>ThreadPool</i> |
|---------------------|--------------------------------|---------------------------------|--------------------------|
| 100 | +0.04 MB | -0.57 MB | +0.01 MB |
| 500 | +0.04 MB | -0.59 MB | +0.01 MB |
| 1000 | +0.04 MB | -0.81 MB | +0.06 MB |
| 5000 | +0.03 MB | +0.15 MB | -0.28 MB |
| 10000 | -0.35 MB | +2.49 MB | +0.16 MB |

Tabela 3 - Operações de I/O

| Nº ficheiros | <i>Modelo de Atores</i> | <i>Thread por tarefa</i> | <i>ThreadPool</i> |
|--------------|-------------------------|--------------------------|-------------------|
| 100 | +0.03 MB | +0.39 MB | +0.01 MB |
| 500 | +0.03 MB | +0.45 MB | -0.36 MB |
| 1000 | +0.04 MB | +0.55 MB | -0.42 MB |
| 5000 | -0.45 MB | +0.76 MB | -0.62 MB |
| 10000 | +0.05 MB | +2.71 MB | -0.41 MB |

Tabela - Numero linhas de código

| | <i>Modelo de Atores</i> | <i>Thread por tarefa</i> | <i>ThreadPool</i> |
|---------------------|-------------------------|--------------------------|-------------------|
| Nº Linhas de Código | 76 | 62 | 58 |



Análise Comparativa

- O Modelo de Atores oferece o melhor desempenho geral e escalabilidade, especialmente para sistemas complexos e I/O intensivo, superando as limitações do GIL e o overhead de threads por tarefa. Utilize ThreadPool para tarefas I/O simples e Thread por Tarefa apenas para protótipos ou cenários muito específicos de baixa concorrência.



Conclusão



Resumo dos Principais



Modelo de Atores: Superior em Eficiência e Escalabilidade. Uso de memória estável, ideal para alta concorrência.



ThreadPool: Adequado para Cargas Médias. Consumo moderado, limitado pelo GIL em I/O.



X Thread por Tarefa: Não Escala Bem. Alto consumo de memória e impacto no sistema.