Processamento sob demanda X Processamento em lote

Atualmente é muito comum realizar e vermos processamento em lote em apis, serviços e funções quaisquer hoje na programação, um exemplo claro pra exemplificar aqui é ler de uma banco de dados 100.000 linhas de usuários e retornar para o cliente todos os 100.000 usuários **carregados em memória**, isso pode ser muito prejudicial pois pode elevar muito o consumo de memória no seu sistema, podendo ocasionar um desperdício de recurso.

Para isso veremos aqui também outro tipo de processamento, o sob demanda, pouco comum, mas quando visto e bem utilizado pode turbinar o uso de apis e serviços em sistemas que podem variar desde pequenos E Commerces até grandes projetos de maior escala, para clarear melhor sobre o'que seria veremos um exemplo, dos mesmos 100.000 usuários do banco, ao invés de pegar todos, carregá-los em memória e enviar como resposta, dessa vez pegamos de pequenas quantidades e vamos enviando e processando de pouquinho em pouquinho, ai é escolha sua processar de 1.000 em 1.000 ou mais ou menos.

A grande vantagem que temos com esse esquema de processamento é a maximização e otimização de recursos, pois agora pegamos de pouco em pouco e processamos e depois de processá-los e retornar ele, em seguidas jogamos fora e limpamos a memória para que os próximos itens sejam pegos e processados.

prós e contras

* **processamento em lote** (PROS):

- processamento massivo e intensivo dos dados, maior velocidade na entrega

* **processamento em lote** (contras):

- consumos altíssimos de memória que poderiam ter sido evitados

- alta chance de bater timeout por causa da memória ainda não liberada

* **processamento sob demanda (**PROS**):**

- processa os dados conforme vai chegando e liberando a memória

- menos uso de memória por requisição ou operação

- abarca mais usuários simultâneos

* **processamento sob demanda (**Contras**):**

- mais lento, portanto responde menos usuário no mesmo tempo em comparação como processamento em lote

Como vimos anteriormente os prós e contras de cada um, agora vamos a prova real, será realizado rounds de testes comparando cada um em tempo médio de resposta, quantidade de memória usada por requisição, quantidade de usuários respondidos com sucesso em determinado tempo, para isso usaremos um servidor node simples, nosso banco de dados seria um array de objetos em memória simples, ferramentas para observar o consumo de memória CLIMEM, e express-status-monitor , e autocannon para fazer o teste de carga;

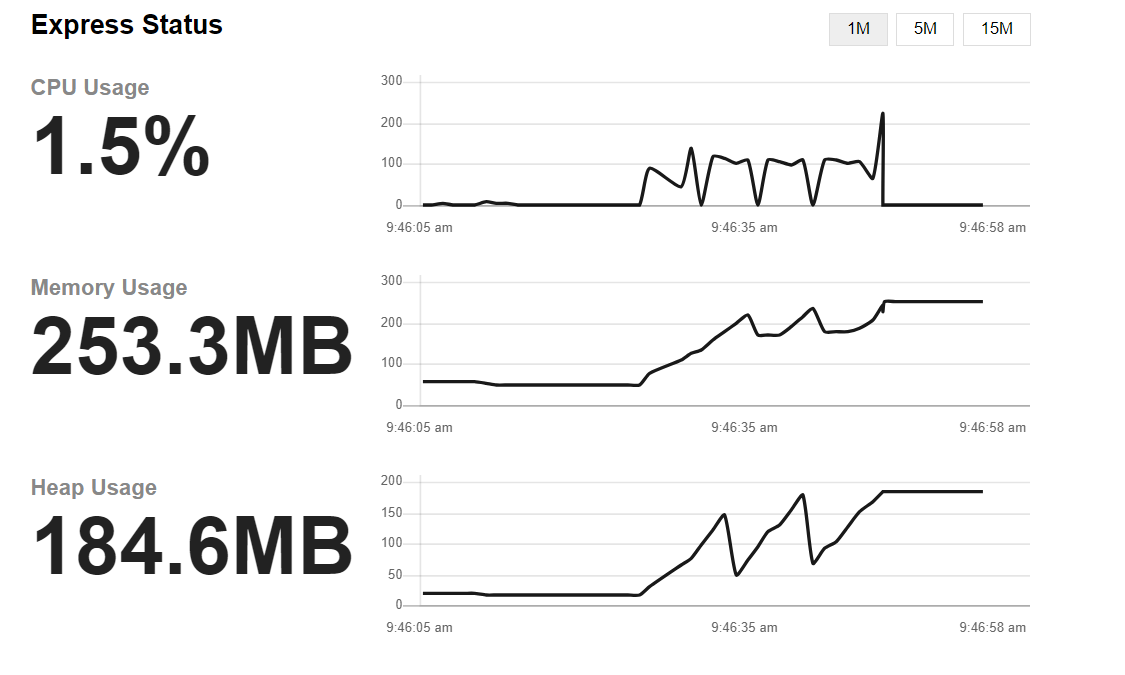
testes: 3000 usuários simultâneos durante 20 segundos

formular usadas:

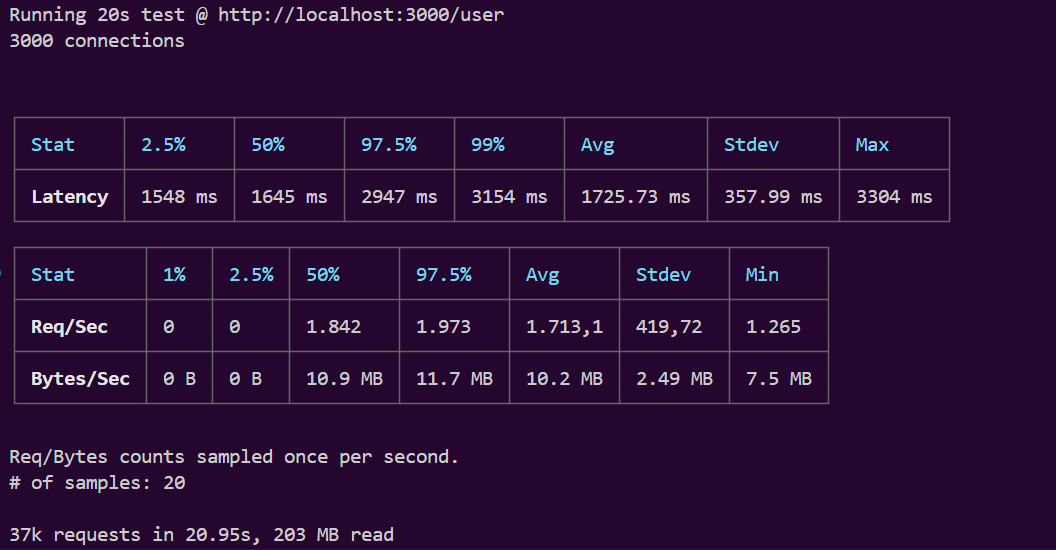
memória por requisição = memória total usada / número de requisições

número de requisições por 100Mb = 100Mb / memória por requisição

## Round 1 Com streams



máximo atingido 250MB



com esse primeiro teste podemos realizar os primeiros cálculos de quantos requisições conseguiram ser feitas a cada 100M,

DADOS APROXIMADOS

memória (alocada) por requisição = 0.006757

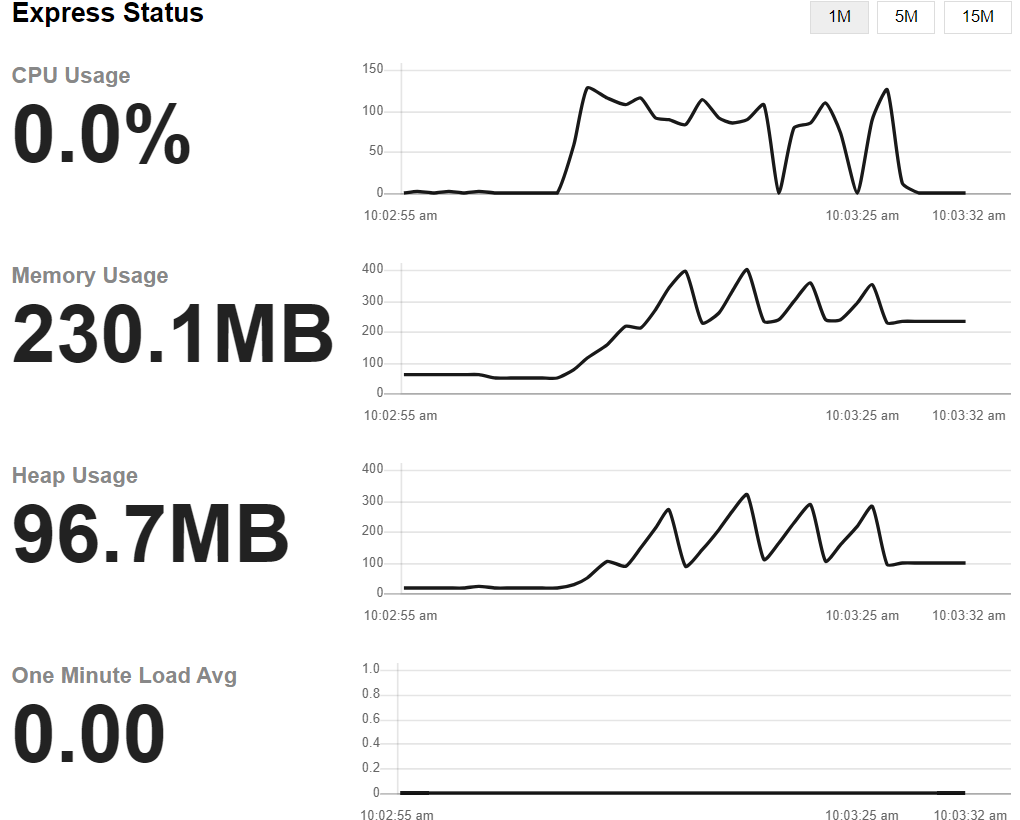
requisições por 100Mb (alocado) = 14800

memória (usada) por requisição = 0.004865

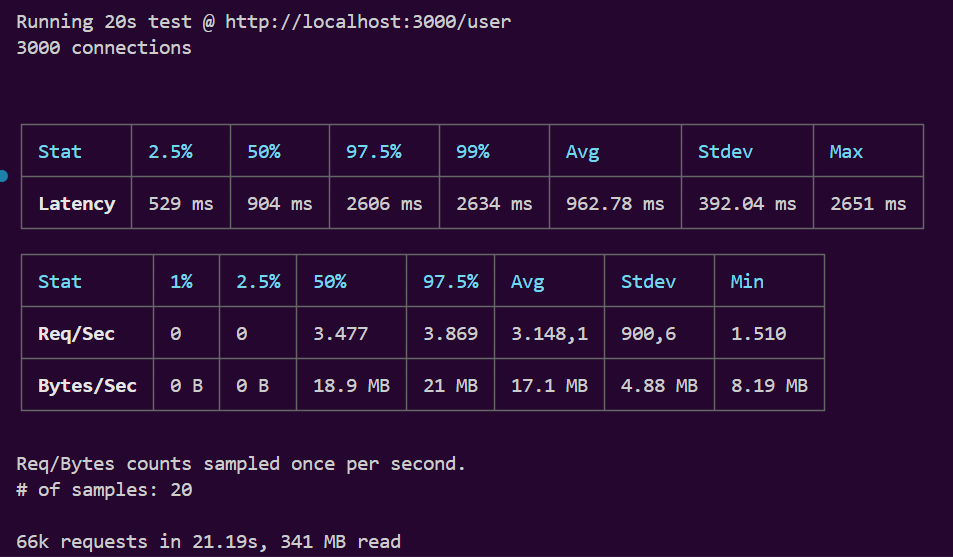
requisições por 100Mb (usada) = 20556

tempo médio de resposta = 1.7 segundos

## Round 1 sem streams



máximo de 400Mb



com esse primeiro teste podemos realizar os primeiros cálculos de quantos requisições conseguiram ser feitas a cada 100M,

DADOS APROXIMADOS

memória (alocada) por requisição = 0.006061

requisições por 100Mb (alocado) = 16500

memória (usada) por requisição = 0.004924

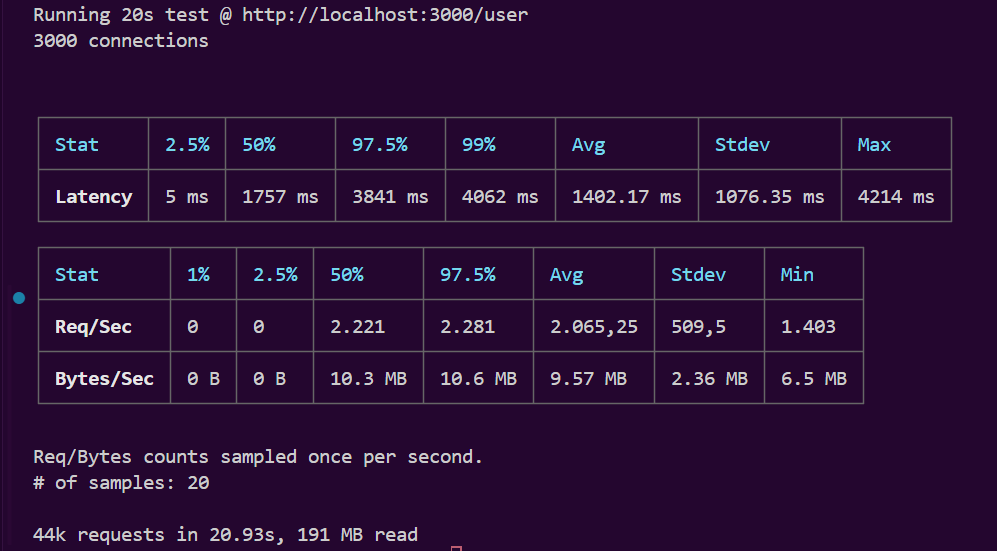
requisições por 100Mb (usada) = 20300

tempo médio de resposta = 960 milissegundos

como vimos aqui nesse primeiro round não tem muita diferença entre os dois, porém o processamento fez mais em menos tempo porém o processamento com streams conseguiu suportar a tarefa ms com mais MB por requisição, de cara alguns vão afirmar “Processamento sob demanda não é mais eficaz que o de lote” mas é aí que muitos não se aprofundam o suficiente para extrair bons resultados.

Agora iremos por Round 2 mas com uma diferença do primeiro, no processamento coms streams vamos empacotar os dados de usuários de 5 em 5 e enviar eles em arrays, e aumentar a capacidade interna dos buffers das streams para elas ter uma maior eficiência e trocar a maneira de como os dados são transmitidos de fluxo para fluxo e trocar o sistema de transmissão de dados entre os fluxos de buffer para strings

## Round 2 com streams

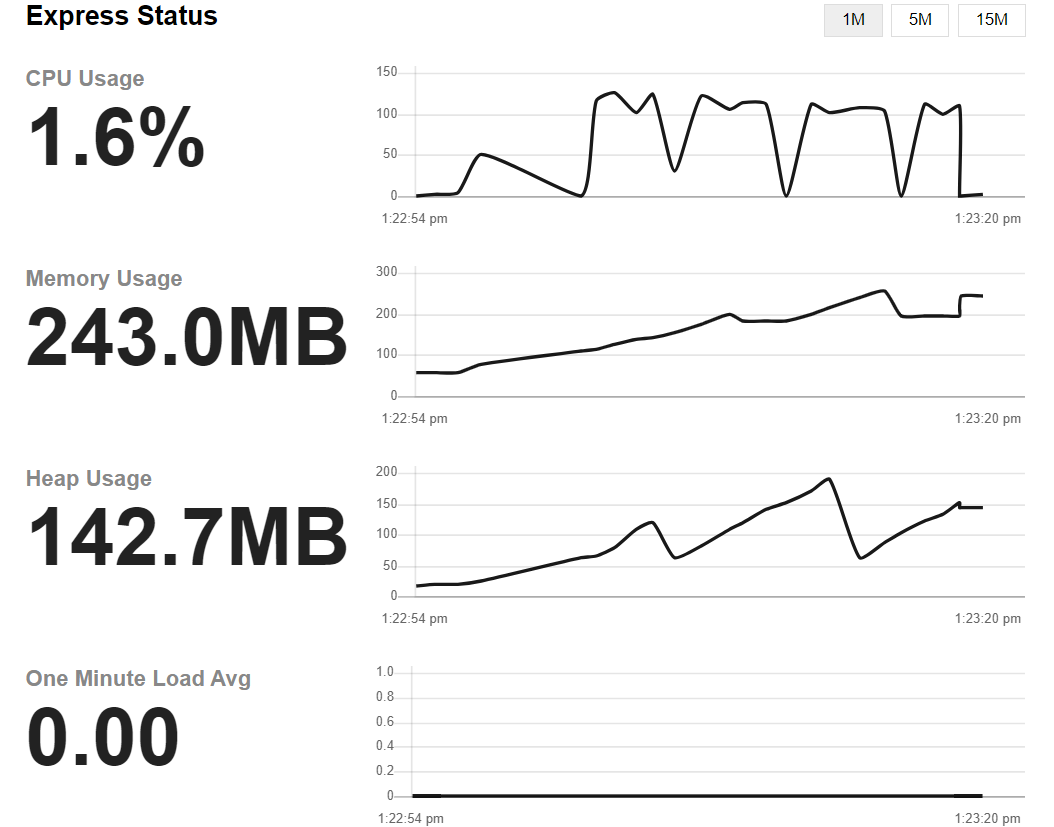


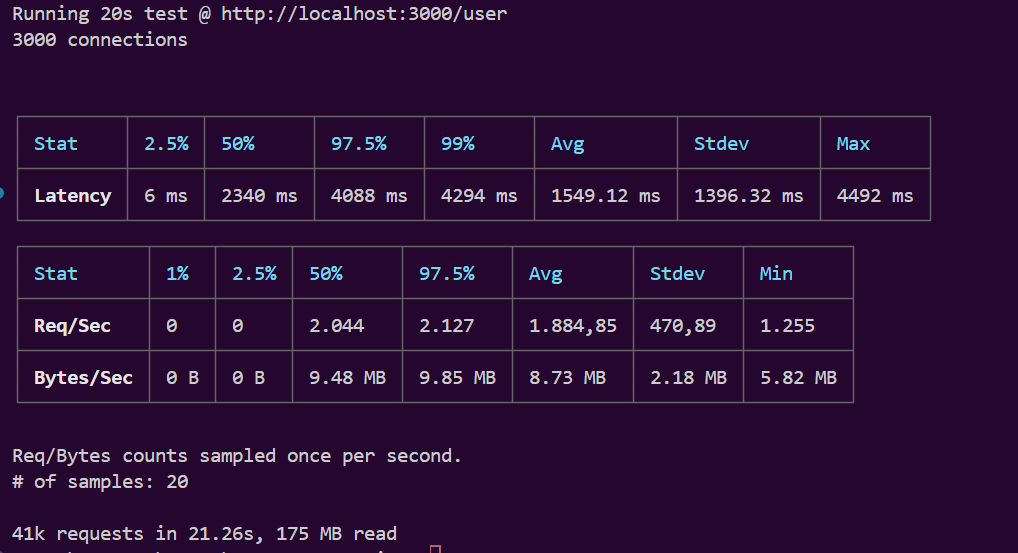
memória consumida 230Mb

memória (alocada) por requisição = 0.005227

requisições por 100Mb (alocado) = 19191

outro teste com streams





memória (alocada) por requisição = 0.005854

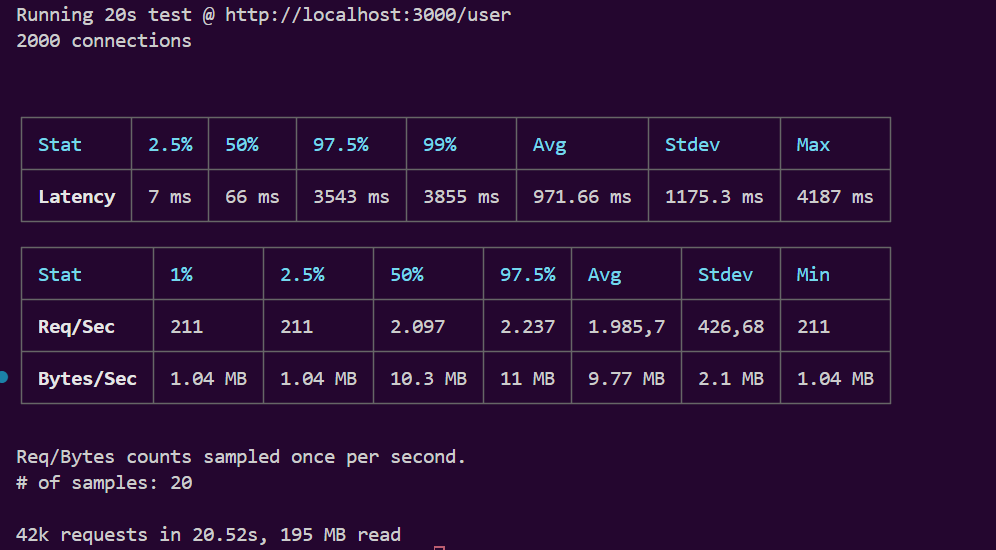
requisições por 100Mb (alocado) = 17081

memória (usada) por requisição = 0.004634

requisições por 100Mb (usada) = 21590

outro teste com streams





memória (alocada) por requisição = 0.005

requisições por 100Mb (alocado) = 20000

memória (usada) por requisição = 0.00333

requisições por 100Mb (usada) = 30000

agora vemos a diferença clara, usando tudo o'que nos é disponibilizado podemos diminuir em muito o consumo de memória das requisições da nossa aplicação,

Conclusão

De fato acabamos de provas que processamento sob demanda usa menos memória que o processamento em lote, ele processa de pequenas em pequenas quantidades, manda pro cliente e pega mais e descarta o'que já usou.

quando devemos usar ou não usar

devemos usar sempre que há uma carga muito grande de dados e uma quantidade de memória limitada que não gostaríamos de ultrapassar, por exemplo você quer fazer um processamento de milhões de relatórios, processe de 100 em 100 ou de acordo, retorne pro cliente, descarte o'que já usou e pegue novos 100.

não devemos substituir o'que já usamos em lote por processamento sob demanda, pois há alguns casos no qual o processamento em lote é melhor, agora que caso é esse? é o caso onde não temos uma grande quantidade de dados, além de deixar ele mais simples, nesses casos ele é mais rápido que processar sob demanda