# SMaRtTrie - Benchmarks

## Sumário



## Experimentos

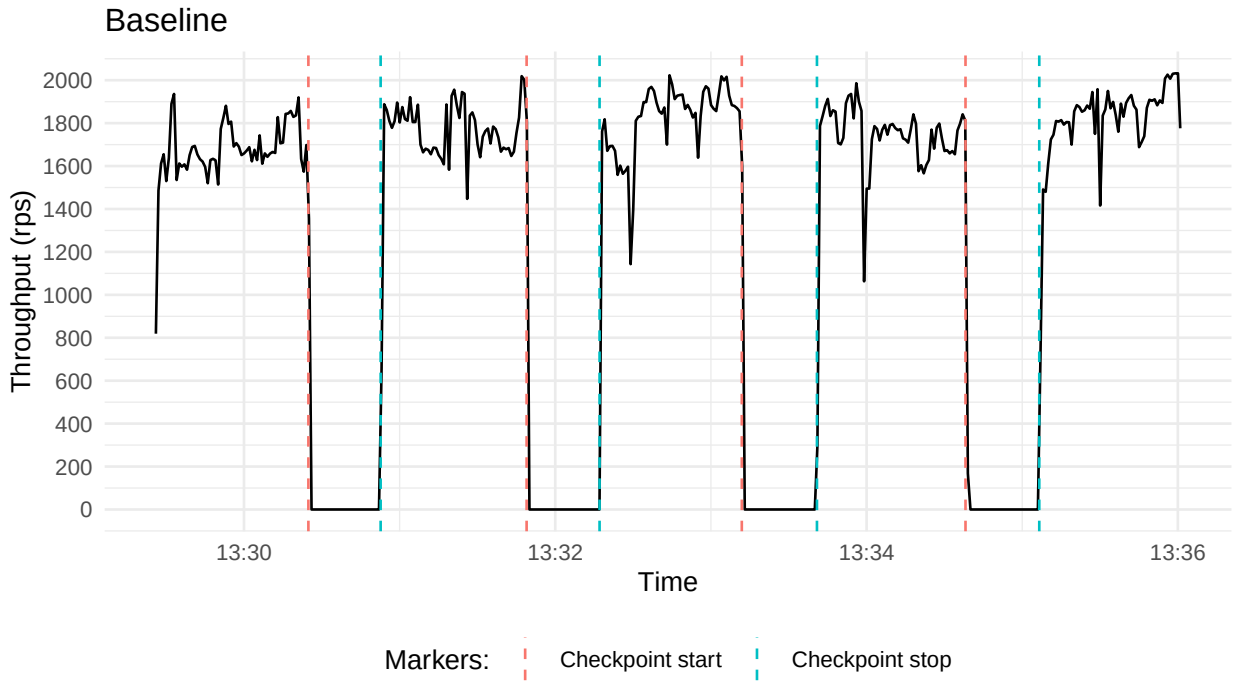
### 2020-10-09 - Baseline

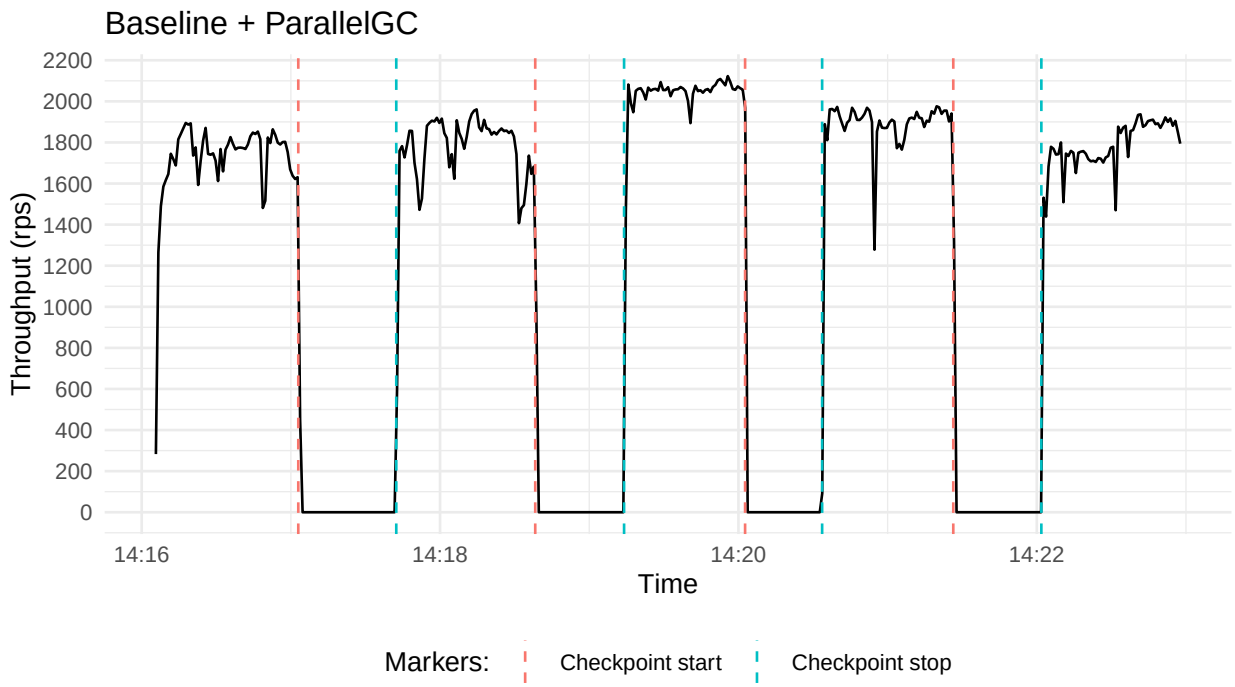
Criação da baseline com aplicação tradicional. Configurações relevantes:

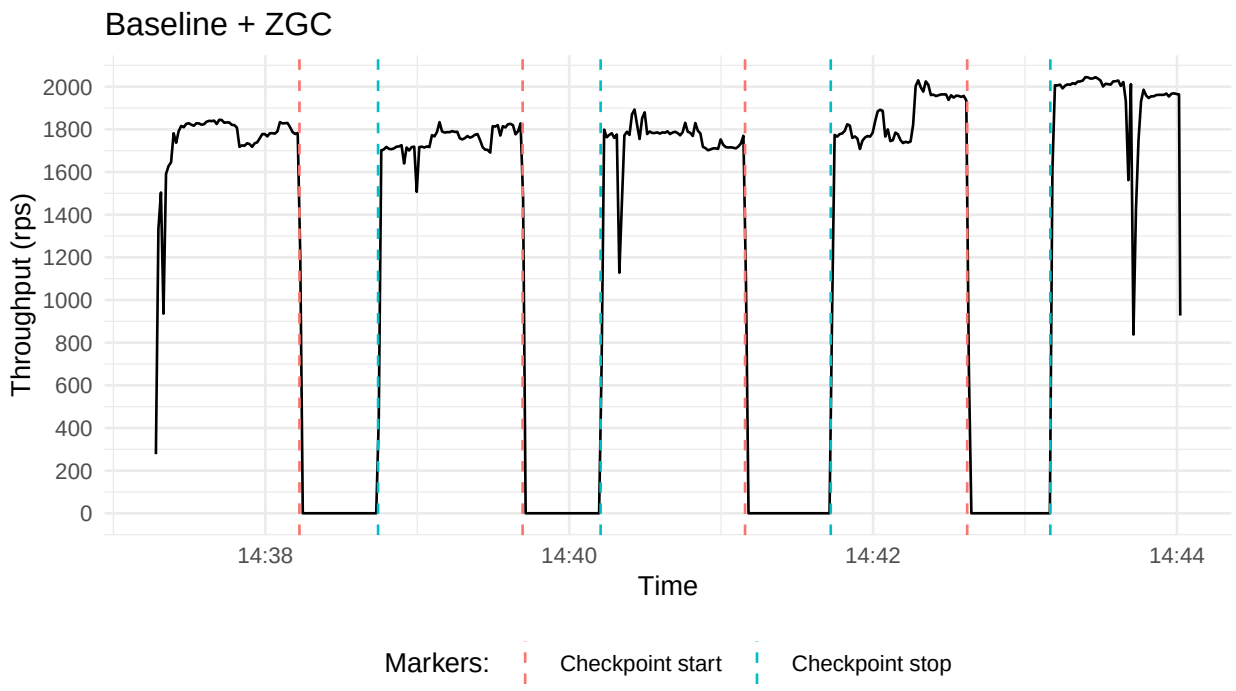
| Configuração | Valor | Observação |
|---|---|---|
| Servidores vs Clientes | 1/1 | Node 91-92 |
| Escrita vs Leitura | 100/0 | |
| Chaves | 500000 | |
| Requests | 500000 | |
| Threads no cliente | 1 | |
| Bytes/Registro | 4kb | |
| Período de checkpoints | 100000 | |
| Checkpoints assíncrono | Não | |
| Log assíncrono | Sim | |
| Estrutura de dados | java.util.TreeMap | |
| Commit | 26544ab | |

Após a carga de dados, o estado em memória atingiu 1967MB (1.92G). Checkpoints levaram entre 30-40

segundos. A vazão da baseline parece irregular nos períodos entre checkpoints, o que motivou a execução com outros tipos de garbage collector. Por padrão a JVM 11 utiliza o algoritmo G1, porém, ZGC aparenta ser o algoritmo mais estável para esta aplicação. Dois bugs foram encontrados: 1) ao tentar executar a aplicação com 1M de chaves, a carga de dados é completada mas, o benchmark tem timeouts em praticamente todas as requisições; 2) ao reiniciar o servidor mantendo os dados persistidos, praticamente todas as requisições resultam em timeout (similar ao problema 1).



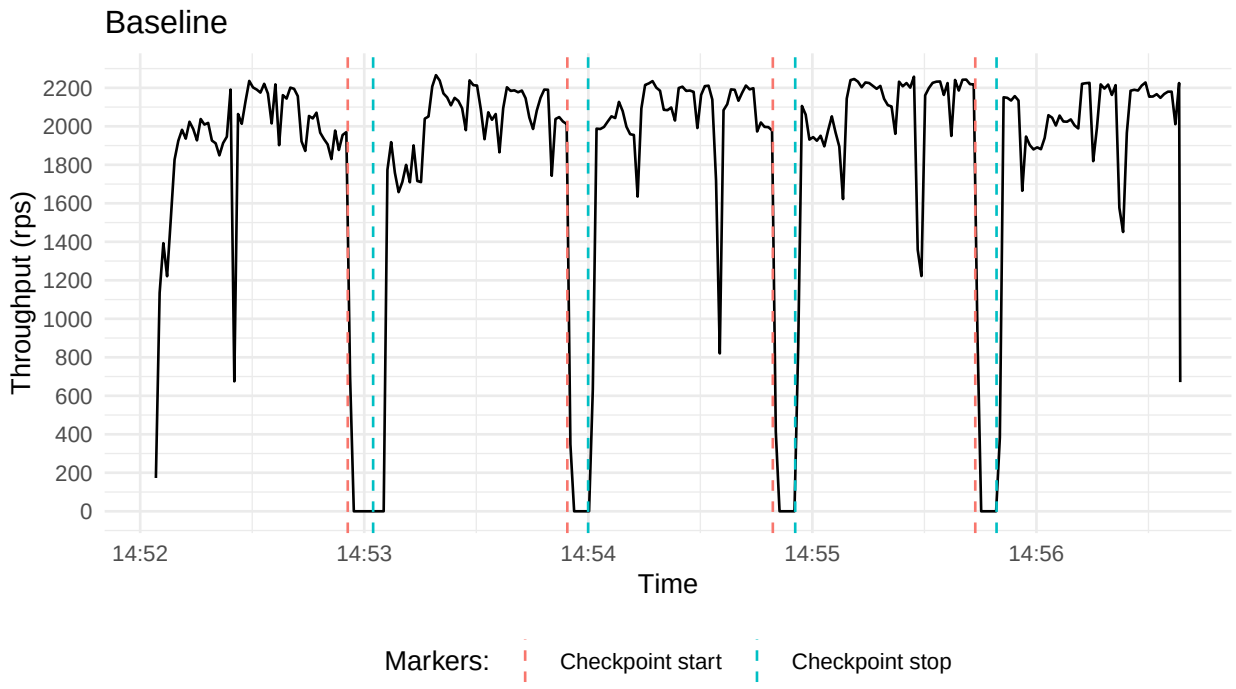



### 2020-10-17 - Reescrita dos Mecanismos de Checkpoint e Log

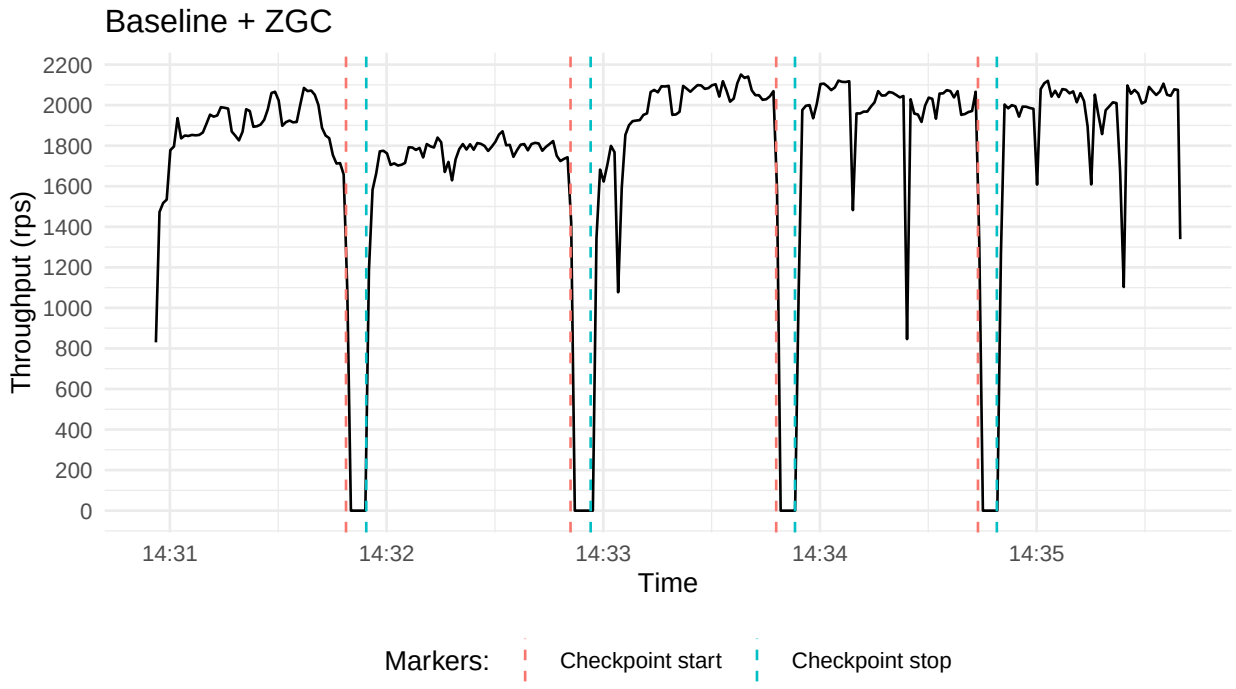
Nesta iteração, assumimos controle dos mecanismos de checkpoint e log. Para avaliar o impacto de tais componentes, reexecutamos a baseline. A seguir estão as configuração relevantes:

| Configuração | Valor | Observação |
|---|---|---|
| Servidores vs Clientes | 1/1 | Node 91-92 |
| Escrita vs Leitura | 100/0 | |
| Chaves | 500000 | |
| Requests | 500000 | |
| Threads no cliente | 1 | |
| Bytes/Registro | 4kb | |
| Período de Checkpoints | 100000 | |
| Checkpoints assíncrono | Sim/Não | TrieMap é assíncrono, demais são síncronos |
| Log assíncrono | Sim | |
| Estrutura de dados | java.util.TreeMap, java.util.concurrent.HashMap, scala.collection.concurrent.TrieMap | |
| Commit | 3d47d90 | |

Iniciamos com a reexecução da baseline, utilizando uma estrutura de dados do tipo Tree-Map, e o algoritmo de GC paralelo. É visível que o tempo de execução do checkpoint diminuiu se comparado com a versão anterior do dia 09, que utiliza os algoritmos do BFT-Smart. No entanto, o comportamento é o mesmo: durante a execução do checkpoint nenhum comando é executado.

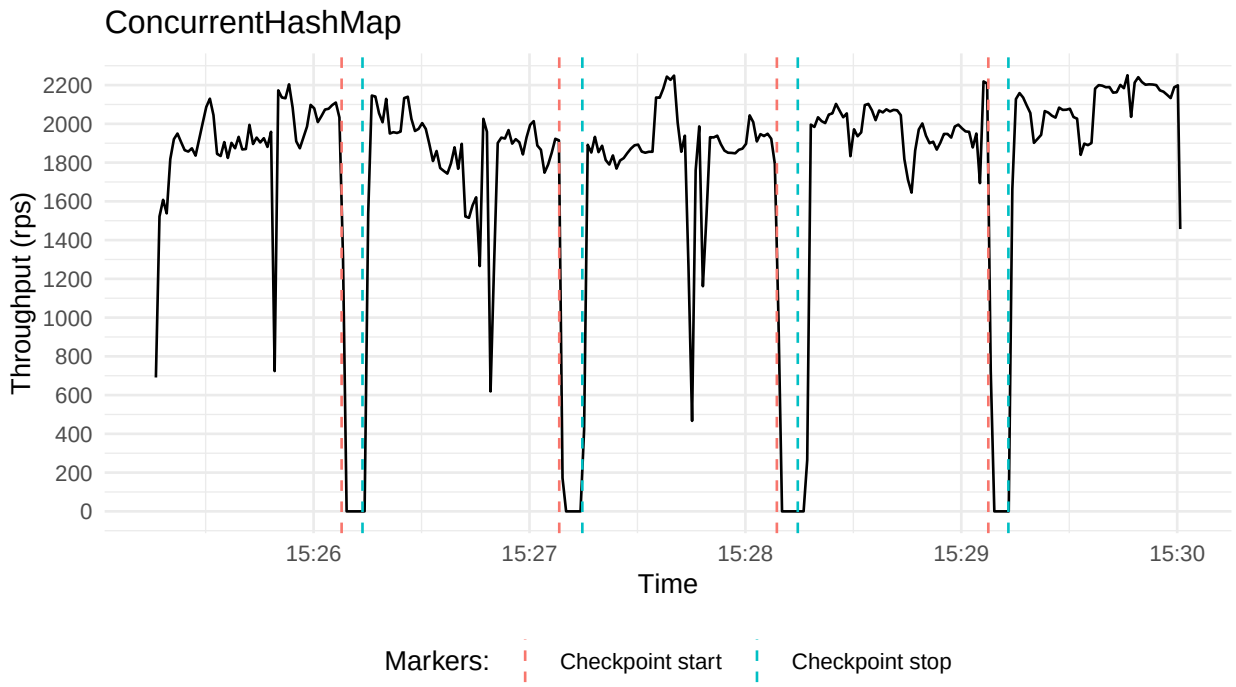



A título de curiosidade, reexecutamos também a baseline com o algoritmo de ZGC de garbage collector. O resultado foi semelhante ao obtido anteriormente: maior estabilidade apensar de menor vazão. A escolha do algoritmo correto de GC ainda não é clara. Os demais experimentos foram executados com o algoritmo paralelo de GC.

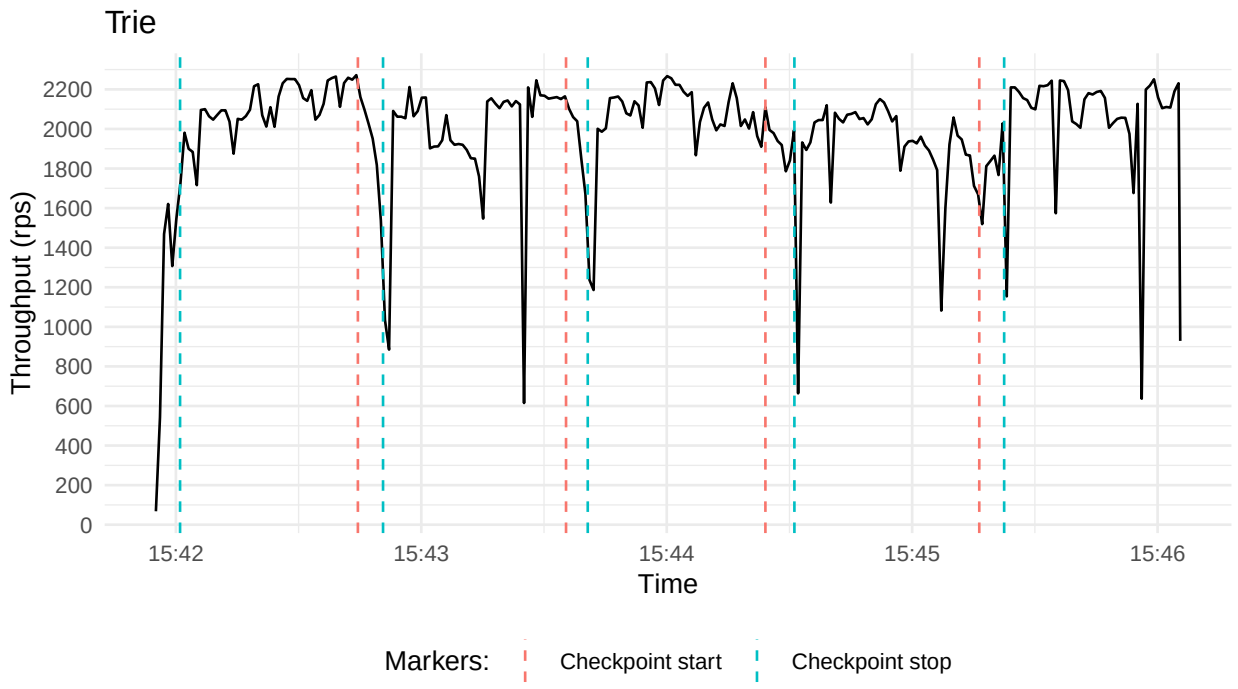



A seguir, executamos um experimento similar, apenas variando o tipo da estrutura de um TreeMap (não thread-safe) para um ConcurrentHashMap (thread-safe). A hipótese é que a vazão de uma estrutura não thread-safe pode ser maior que a de uma estrutura thread-safe e, portanto, não seria uma comparação justa

com a estrutura thread-safe CTrie. Ambas estruturas variam na mesma faixa de ~1800 a ~2200 RPS, sendo que a TreeMap se mantém mais próximo ao limite superior. Apesar disto, a instabilidade na vazão do serviço em ambos os casos torna difícil chegar a qualquer conclusão.




Finalmente, executamos o mesmo experimento com uma estrutura do tipo CTrie, ou TrieMap em scala. Apesar da vazão ainda instável, é possível notar que não há a interrupção total do serviço durante o checkpoint. É interessante observar que ao final dos checkpoints, geralmente ocorre uma grande queda na vazão. Acreditamos que a queda esteja relacionada a uma rotina sincronizada que "trunca" os logs ao final do checkpoint.



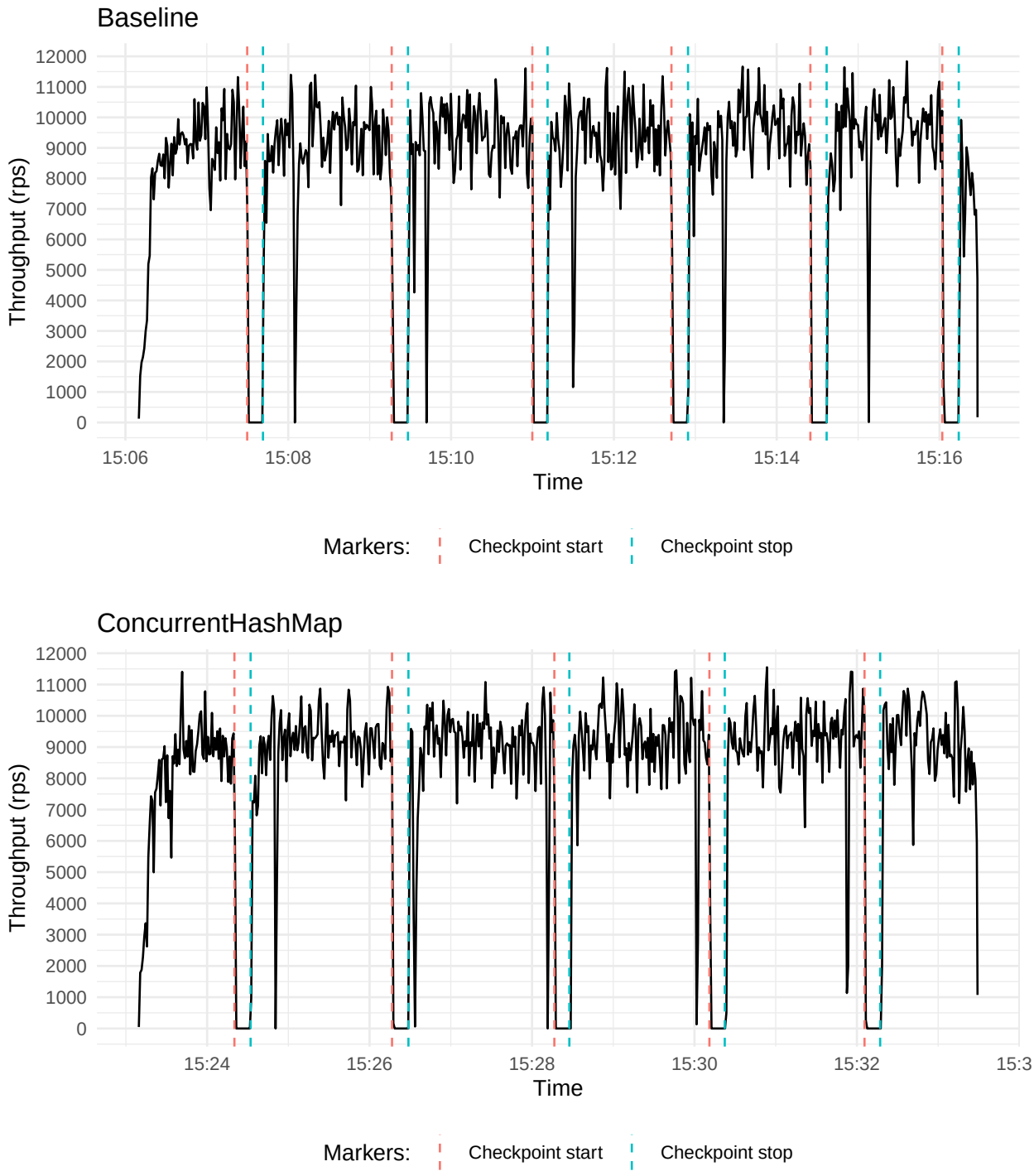

O experimento com a CTrie se mostra promissor. No entanto, a instabilidade na vazão do serviço dificulta a sua análise. Próximos passos:

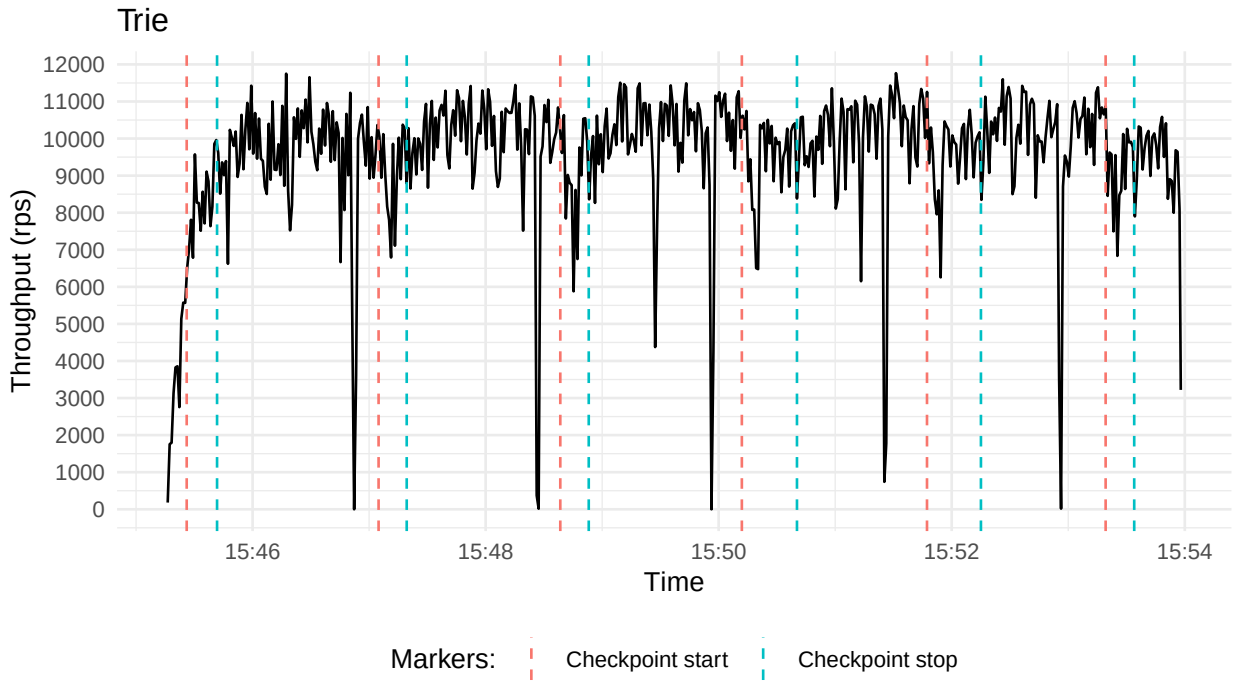
- Depurar o código afim de estabilizar a vazão do serviço;
- Reexecutar os experimentos.

### 2020-10-23 - Otimizações de memória e carga de dados

| Configuração | Valor | Observação |
|---|---|---|
| Servidores vs Clientes | 1/1 | Node 41-42 |
| Escrita vs Leitura | 100/0 | |
| Chaves | 500000 | |
| Requests | 5000000 | |
| Threads no cliente | 32 | |
| Bytes/Registro | 4kb | |
| Período de Checkpoints | 100000 | |
| Checkpoints assíncrono | Sim/Não | TrieMap é assíncrono, demais são síncronos |
| Log assíncrono | Sim | |
| Estrutura de dados | java.util.TreeMap, java.util.concurrent.HashMap, scala.collection.concurrent.TrieMap | |
| Commit | c4cafdc | |

Neste experimento buscamos avaliar as otimizações feitas para reduzir as pausas de GC durante a execução dos sistema. Além disso, verificamos que os servidores 91 e 92 possuem um número grande de processos parallelos a execução do experimento (aproximadamente 700). Utilizamos, então, os servidores 41 e 42 que possuem memos processos em execução (~130). Aumentamos também o número de theads do benchmark afim de evitar períodos de lock no servidor devido a falta de trabalho a ser executado.





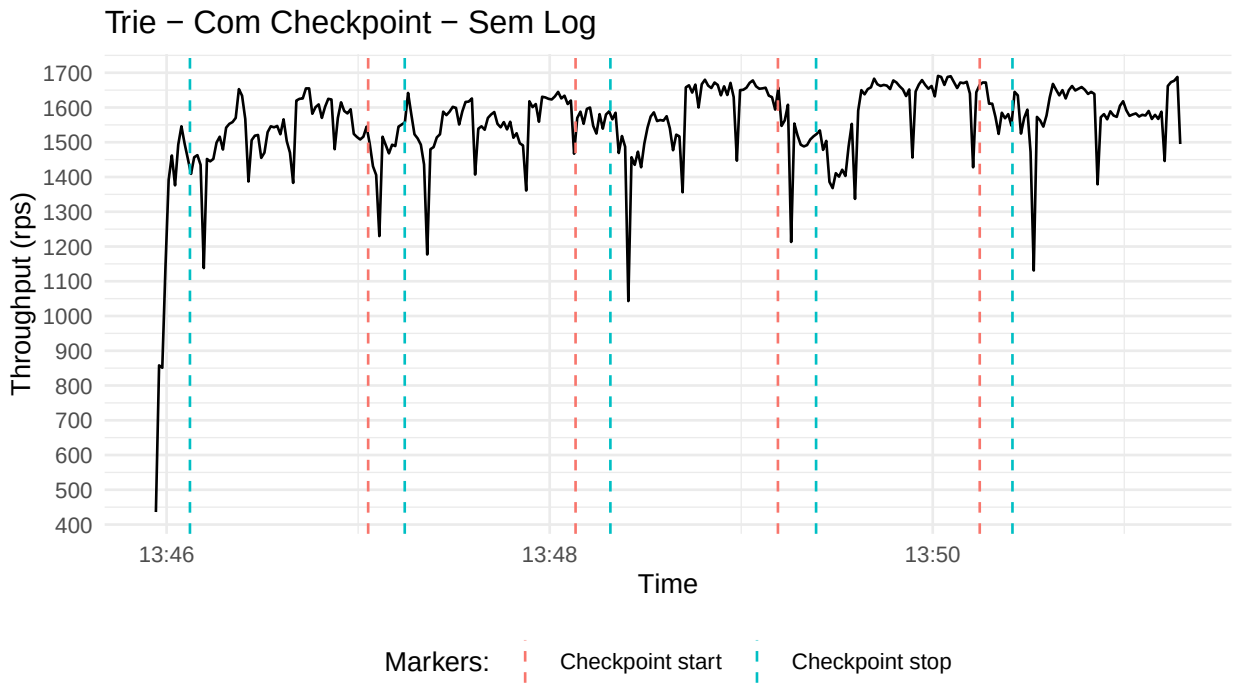

Apesar do aumento na vazão, a tempo de serviço continua instável. A variação aumentou em realção aos experimentos ateriores, o que dificulta ainda mais as análises.

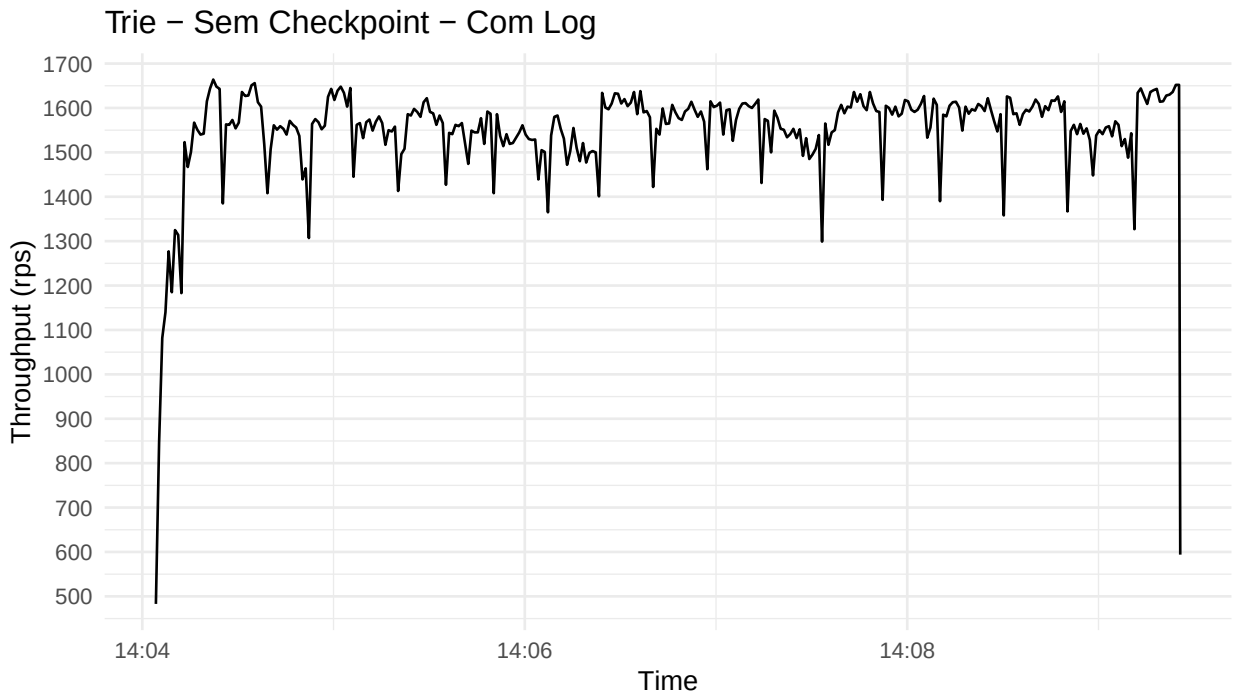
### 2020-11-05 - Experimentos sem checkpoint e sem log

| Configuração | Valor | Observação |
|---|---|---|
| Servidores vs Clientes | 1/1 | Node 41-42 |
| Escrita vs Leitura | 100/0 | |
| Chaves | 500000 | |
| Requests | 500000 | |
| Threads no cliente | 1 | |
| Bytes/Registro | 4kb | |
| Período de Checkpoints | 100000 | |
| Checkpoints assíncrono | Sim | |
| Log assíncrono | Sim | |
| Estrutura de dados | scala.collection.concurrent.TrieMap | |
| Commit | 686eb73 | Alterações foram feitas no código de maneira experimental (não há commits para cada experimento). |

Para avaliar o impacto dos mecanismos de checkpoint e log na vazão e aparente instabilidade do serviço. Os seguintes experimentos foram executados removendo as rotinas de log e checkpoint.


Trie − Sem Checkpoint − Sem Log
Throughput (rps)
1700
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
13:20
13:22
13:24
Time


Trie − Com Checkpoint − Sem Log
Throughput (rps)
1700
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
500
400
13:46
13:48
13:50
Time
Markers:
Checkpoint start
Checkpoint stop



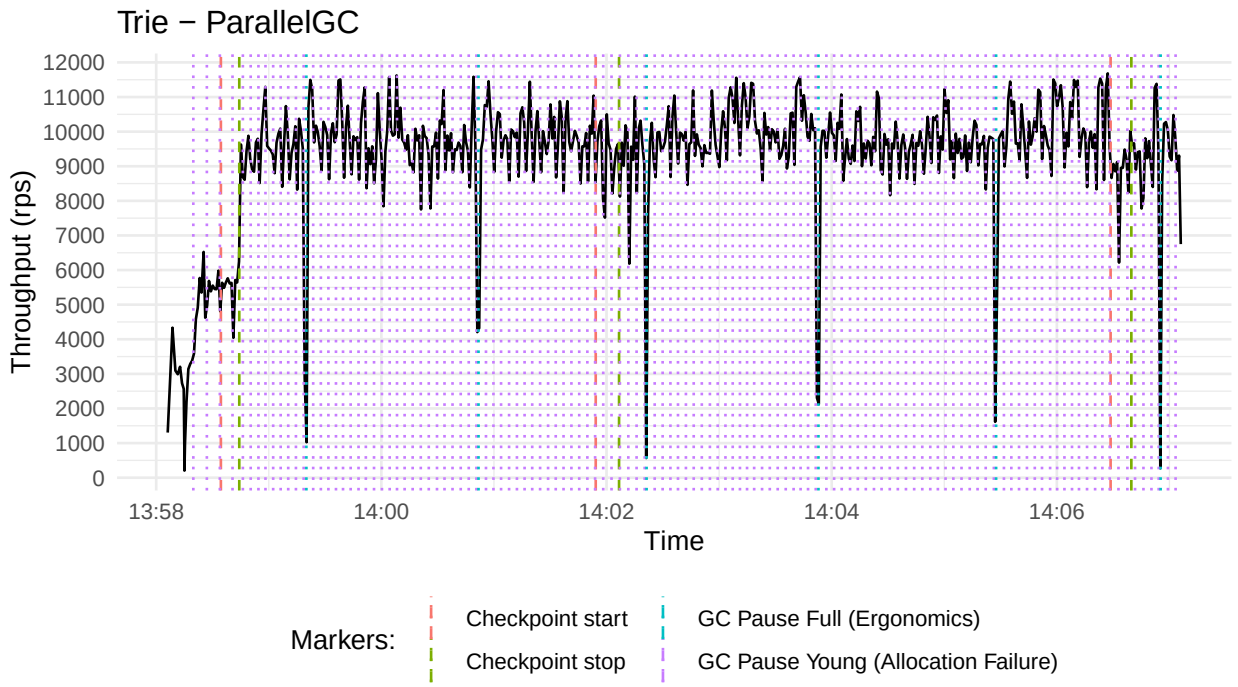

É interessante perceber que mesmo sem checkpoints e logs, ainda ocorreu um período em que a vazão caiu a zero. O mesmo período não se repetiu nos experimentos seguintes, o que pode indicar que este seja um fator do ambiente, como pausas para GC ou perca de prioridade do SO. No entando, períodos de menor vazão parecem coinsidir com o término de checkpoints. Uma hipótese seria que o snapshot da CTrie criada em memória durante o checkpoint é descartado neste momento, liberando um espaço de memória significativo, o que pode disparar uma pausa de GC. Acredito que podemos remover o log a fim de reduzir as variáveis que podem impactar a vazão. Seria interessnate executar experimentos maiores, coletando uma média de vazão por alguns segundos, reduzindo assim a variação e focando as análises no seu valor médio ao longo do tempo.

### 2020-11-12 - Análize do impacto dos coletores de lixo

| Configuração | Valor | Observação |
|---|---|---|
| Servidores vs Clientes | 1/1 | Node 41-42 |
| Escrita vs Leitura | 100/0 | |
| Chaves | 500000 | |
| Requests | 5000000 | |
| Threads no cliente | 64 | |
| Bytes/Registro | 4kb | |
| Período de Checkpoints | 100000 | |
| Checkpoints assíncrono | Sim | |
| Log assíncrono | - | Log desabilitado |
| Estrutura de dados | scala.collection.concurrent.TrieMap | |
| Commit | 20110f9 | |

Neste experimento, removemos a rotina de log a fim de minimizar o número de variáveis que possam alterar a vazão. Comparamos em detalhe a execução do sistema com diferentes algoritmos de coleção. O primeiro experimento foi executado com o algoritmo ParallelGC, onde é possível observar quedas expressivas na vasão

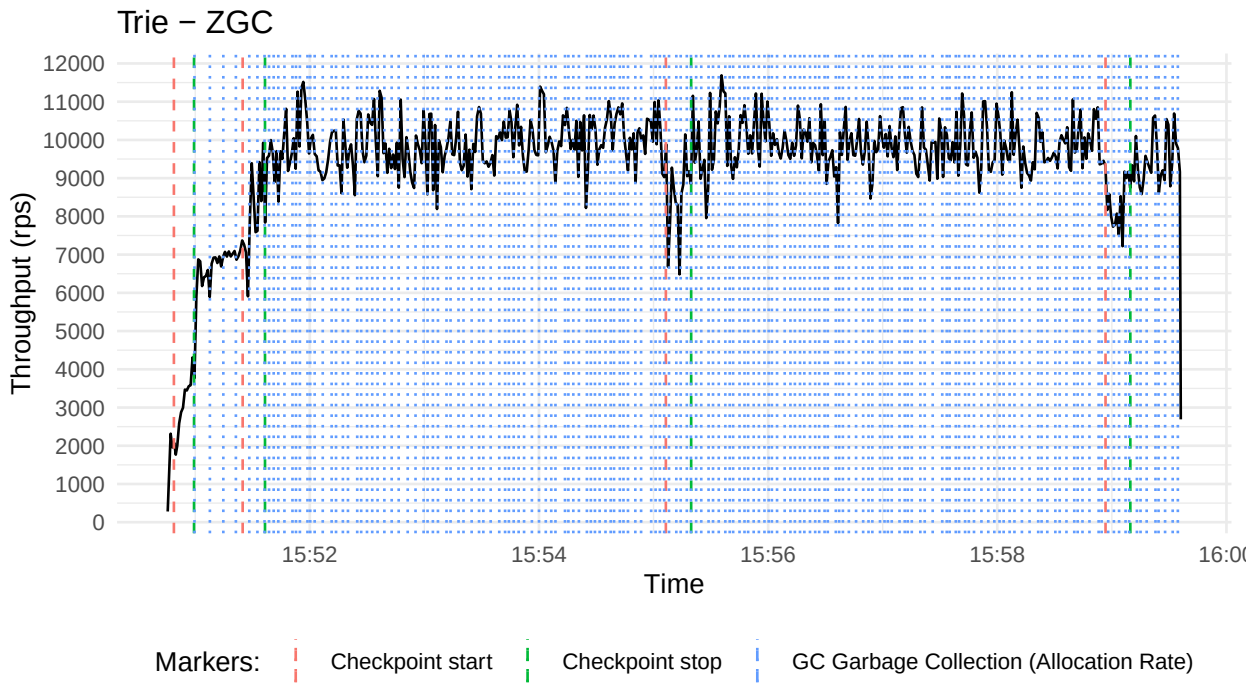
do serviço. Ao cruzar a vazão com as pausas de GC, é possível perceber que grandes quedas estão relacionadas a pausas longas do tipo “GC Pause Full”.



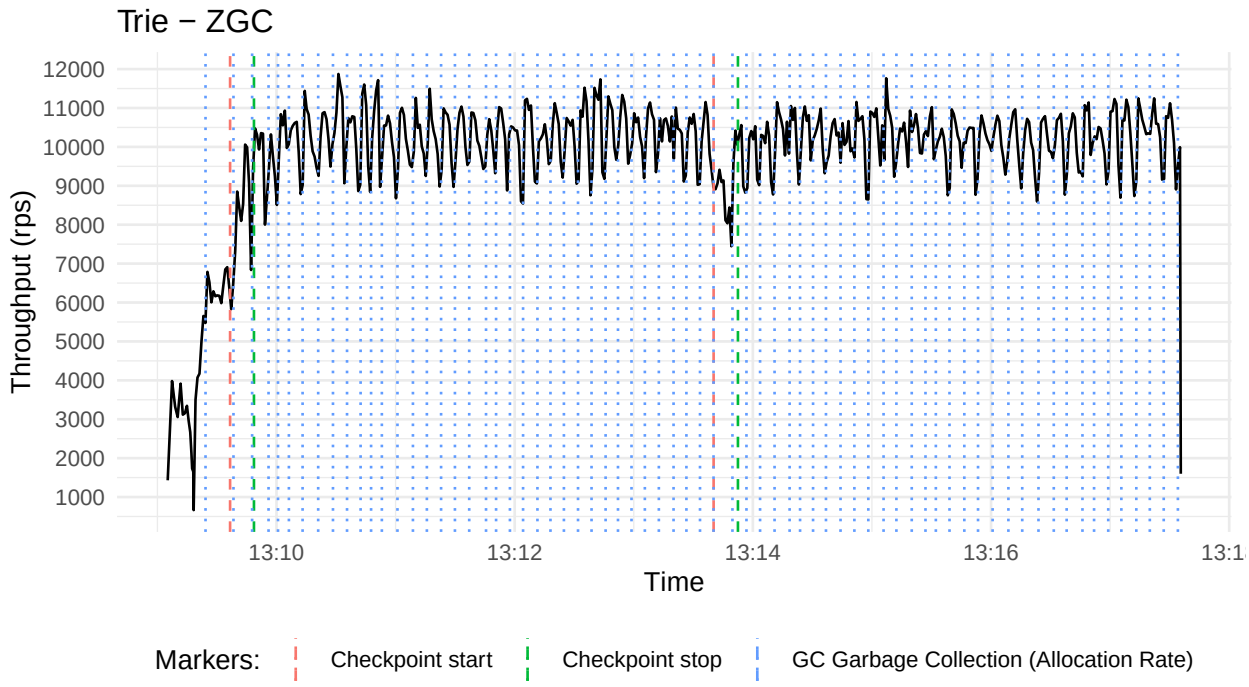

No experimento seguinte, avaliamos o desempenho ao escolher o algoritmo G1. É possível notar que este algoritmo passa por ciclos compostos de pequenas pausas que se tornam cada vez mais frequetes até um determinado limite onde pausas mais expressivas de GC são executadas o que resulta em períodos de menor vazão, ao contrário dos picos de baixa vazão encontrados no algoritmo ParallelGC.


Finalmente, re-executamos o mesmo experimento com o algoritmo ZGC. Este algoritmo apresenta pausas

frequentes, o que resulta em uma maior desvio padrão, porém não há momentos de queda expressiva na vazão se comparado com os algoritmos anteriores.

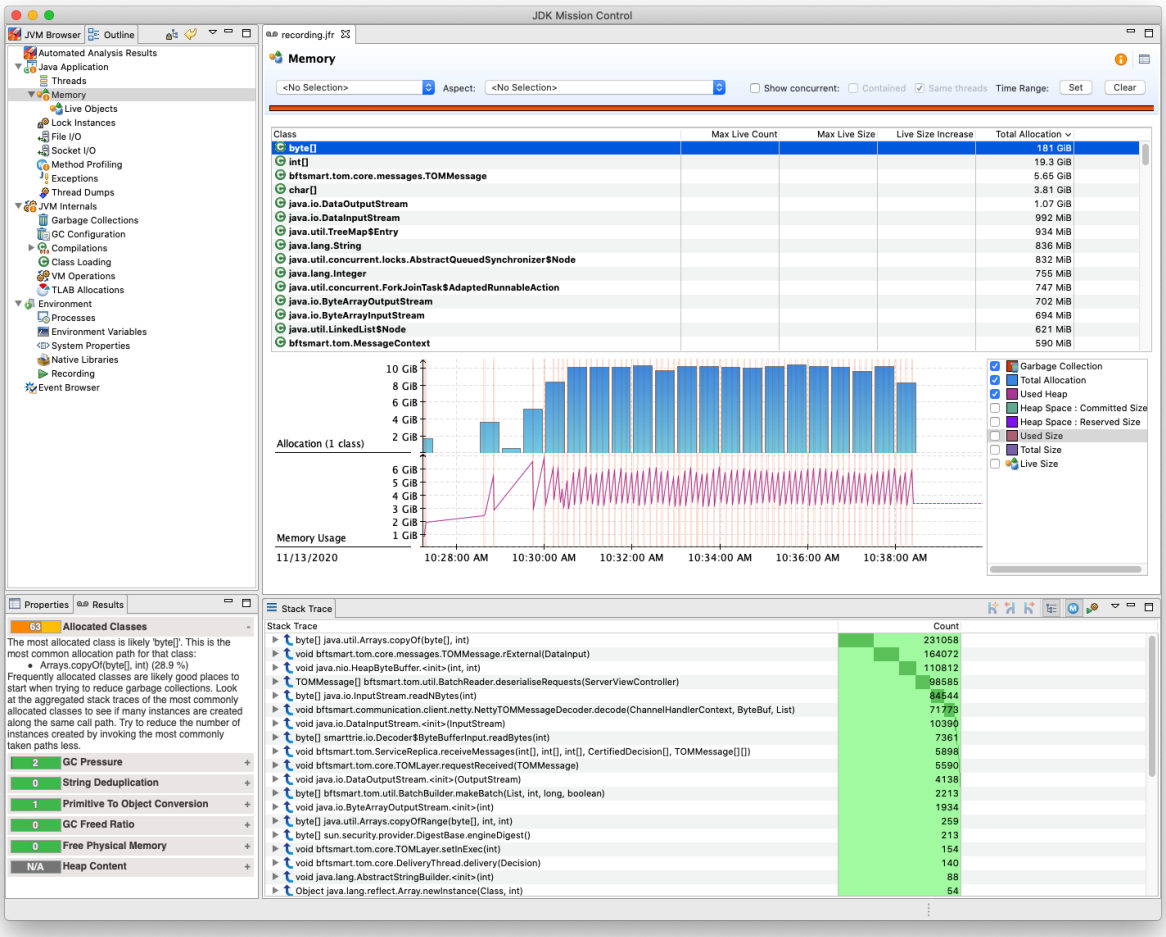



Ao aumentar o tamanho máximo da memória de 6G para 8G, o ZGC preserva o desvio padrão, mas aumenta o periodo entre GCs.




Java flight recorder aponta para arrays de bytes como maior fonte de alocação. BFT-Smart, ao adotar arrays de bytes como estrutura base para representação de dados acaba por colocar muita pressão nos coletores de lixo uma vez que esta estrutura não pode ser reutilizada. A re-engenharia do BFT para utilizaar byte buffers

reutilizáveis (pooled) pode amenizar este problema.

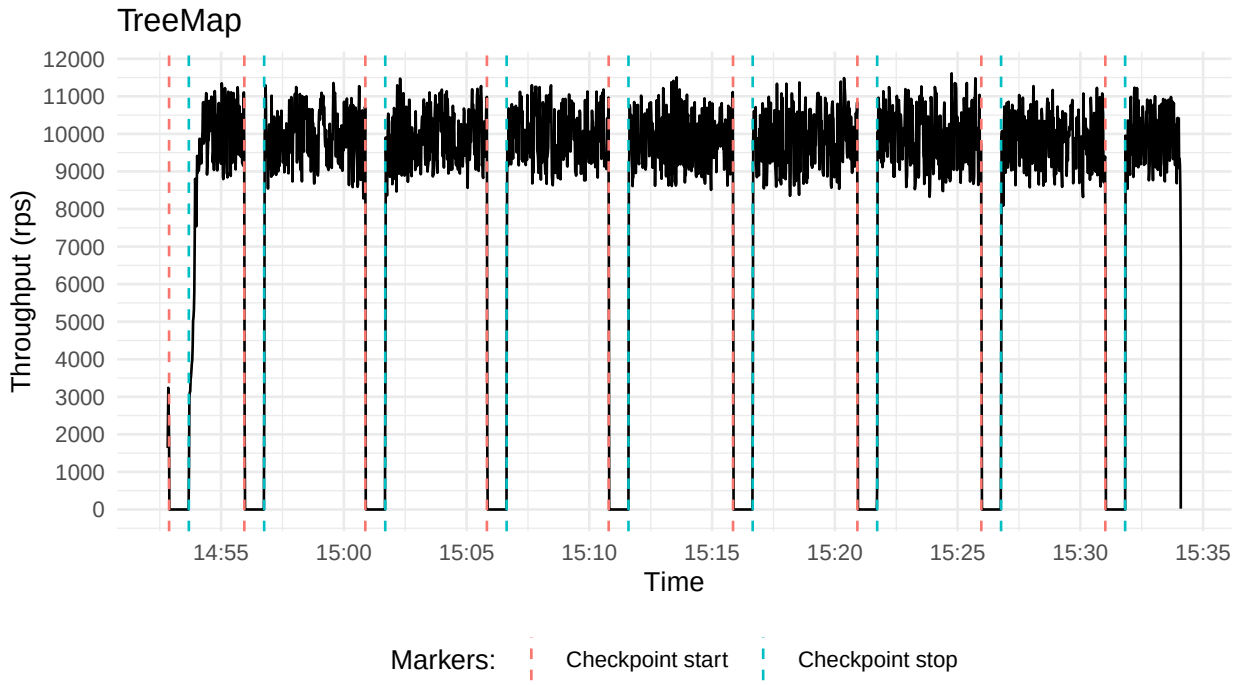


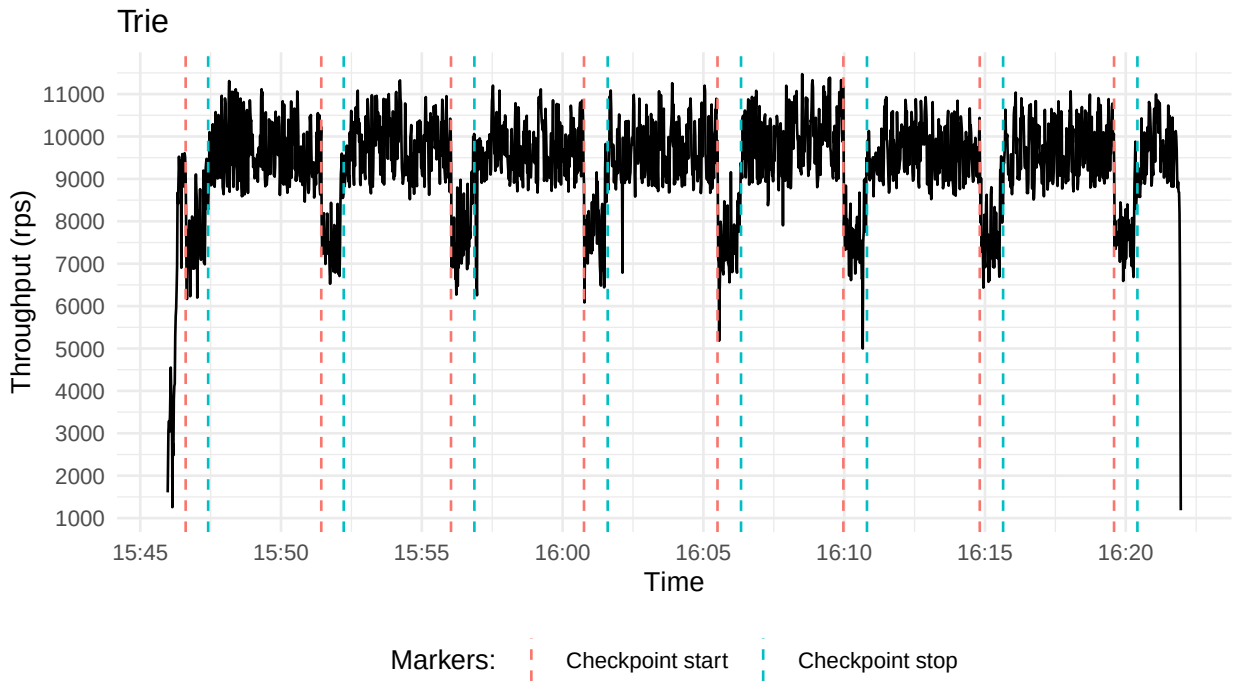

## 2020-11-19 - Experimentos de longa duração

| Configuração | Valor | Observação |
|---|---|---|
| Servidores vs Clientes | 1/1 | Node 41-42 |
| Escrita vs Leitura | 100/0 | |
| Chaves | 1500000 | |
| Requests | 20000000 | |
| Threads no cliente | 64 | |
| Bytes/Registro | 4kb | |
| Período de Checkpoints | 100000 | |
| Checkpoints assíncrono | Sim | |
| Log assíncrono | - | Log desabilitado |
| Estrutura de dados | java.util.TreeMap e scala.collection.concurrent.TrieMap | |
| Commit | ce0d34d | |

Re-executamos os experimentos anteriores por maior duração a fim de coletar dados suficientes para escolher recortes que demostrem a execução dos sitema com vários checkpoints sendo criados. Para tal fim, aumen-

tamos o tamanho do dataset para 5G e executamos 20M de requests variando a estrutura de dados. Nestes experimentos utilizamos o coletor de lixo ZGC com uma heap de 16G.

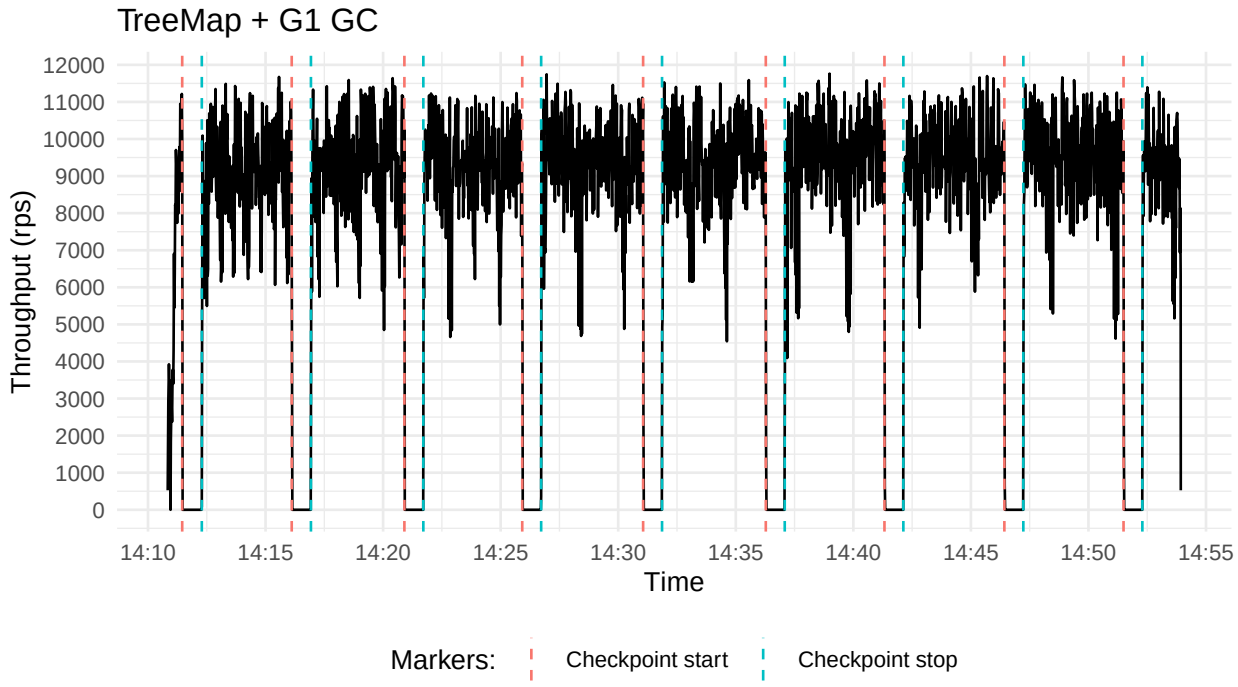


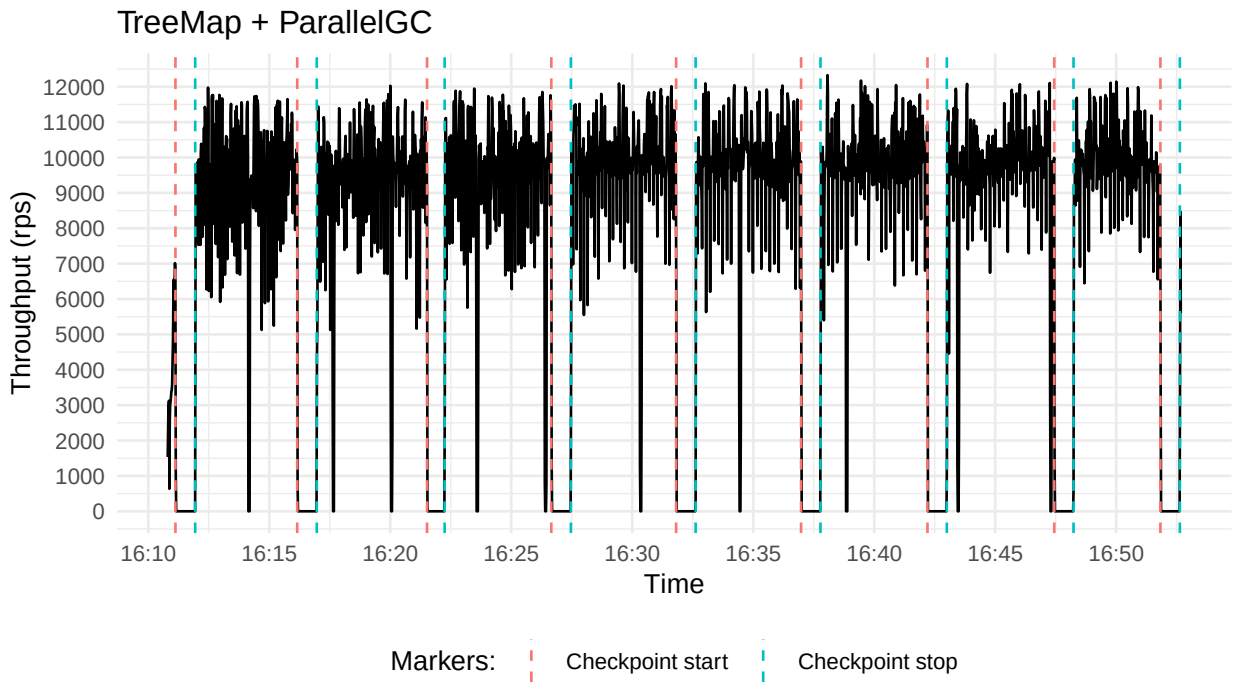




### 2020-12-07 - Coleta de dados de GC para o artigo final

| Configuração | Valor | Observação |
|---|---|---|
| Servidores vs Clientes | 1/1 | Node 41-42 |
| Escrita vs Leitura | 100/0 | |
| Chaves | 1500000 | |
| Requests | 20000000 | |
| Threads no cliente | 64 | |
| Bytes/Registro | 4kb | |
| Período de Checkpoints | 100000 | |
| Checkpoints assíncrono | Sim | |
| Log assíncrono | - | Log desabilitado |
| Estrutura de dados | java.util.TreeMap e scala.collection.concurrent.TrieMap | |
| Commit | ce0d34d | |

Repetimos o experimento anterior apenas coletando dados de GC da baseline utilizando G1 e Parallel GC para fins de comparação no artigo final.

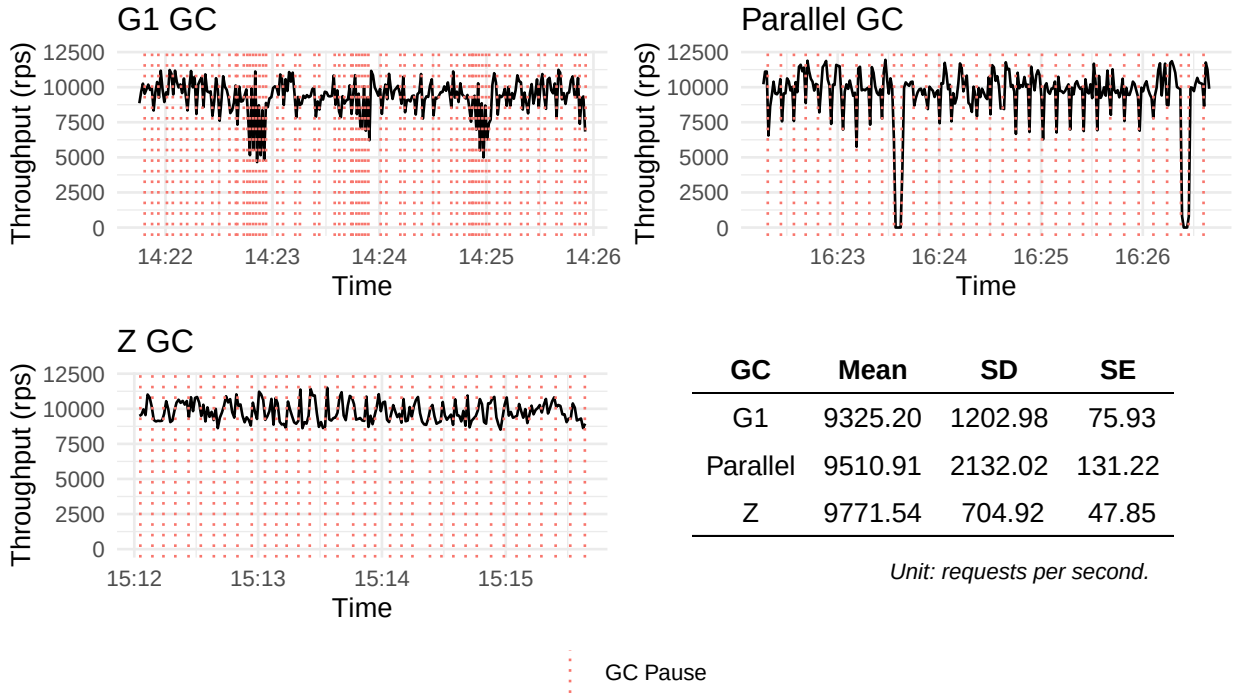





## 2021-01-31 - Análise estatística

*Não houveram novos experimentos.*

**Vazão em relação ao algoritmo de GC utilizado.**




| GC | Mean | SD | SE |
|---|---|---|---|
| G1 | 9325.20 | 1202.98 | 75.93 |
| Parallel | 9510.91 | 2132.02 | 131.22 |
| Z | 9771.54 | 704.92 | 47.85 |

*Unit: requests per second.*

**Vazão média em relação ao modelo de concorrência**

| Model | Mean | SD | SE |
|---|---|---|---|
| Blocking | 8070.79 | 3810.14 | 76.54 |
| Concurrent | 9259.26 | 1202.23 | 25.87 |
| - | 14.73 | -68.45 | -66.20 |

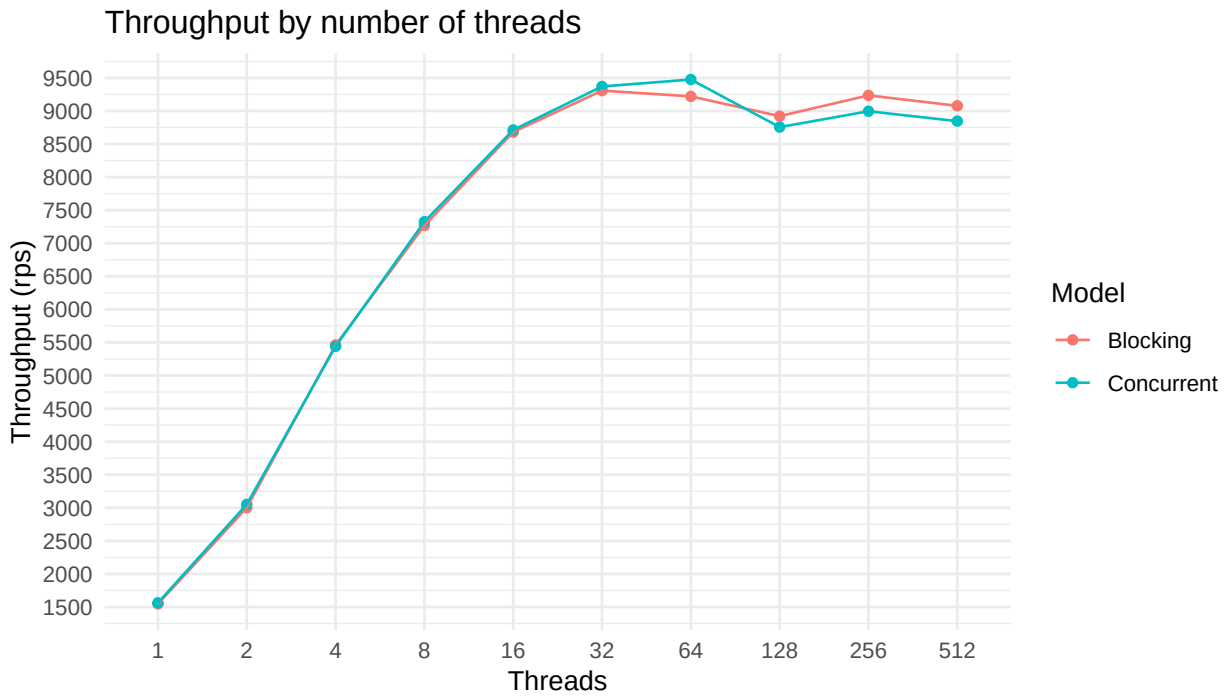
**Vazão média fora e dentro de checkpoints**

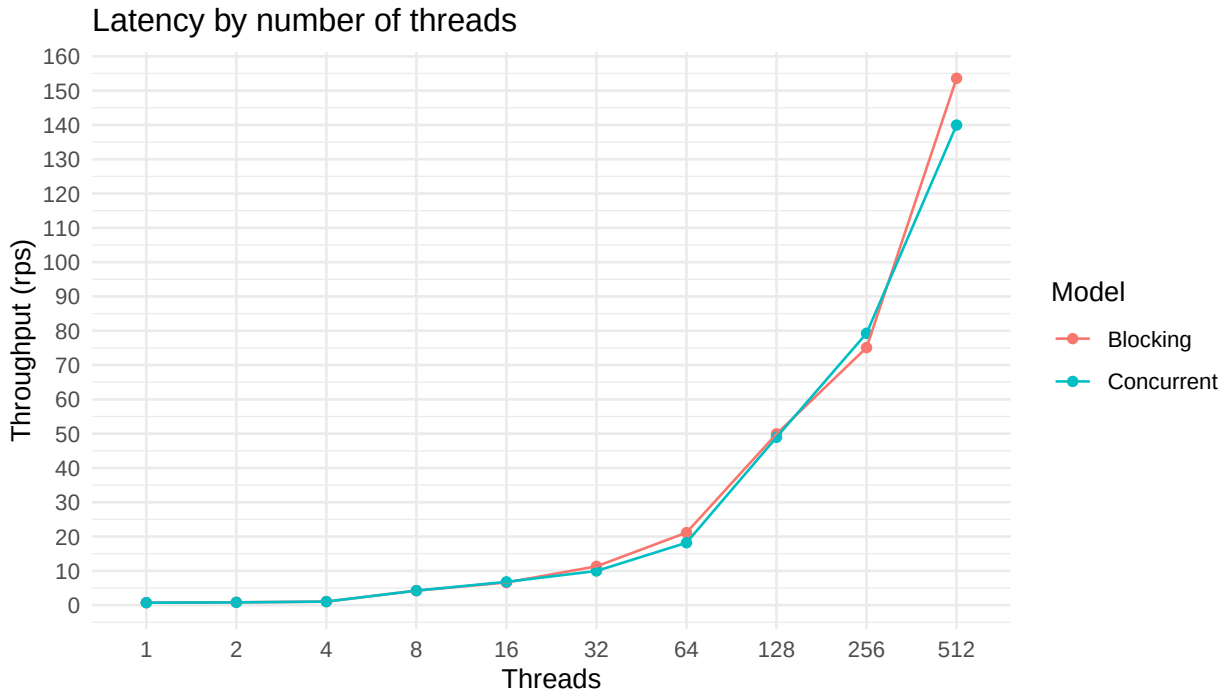
| Checkpoint | Mean | SD | SE |
|---|---|---|---|
| Out | 9605.07 | 984.08 | 23.42 |
| In | 7714.06 | 809.33 | 40.72 |
| - | -19.69 | -17.76 | 73.85 |

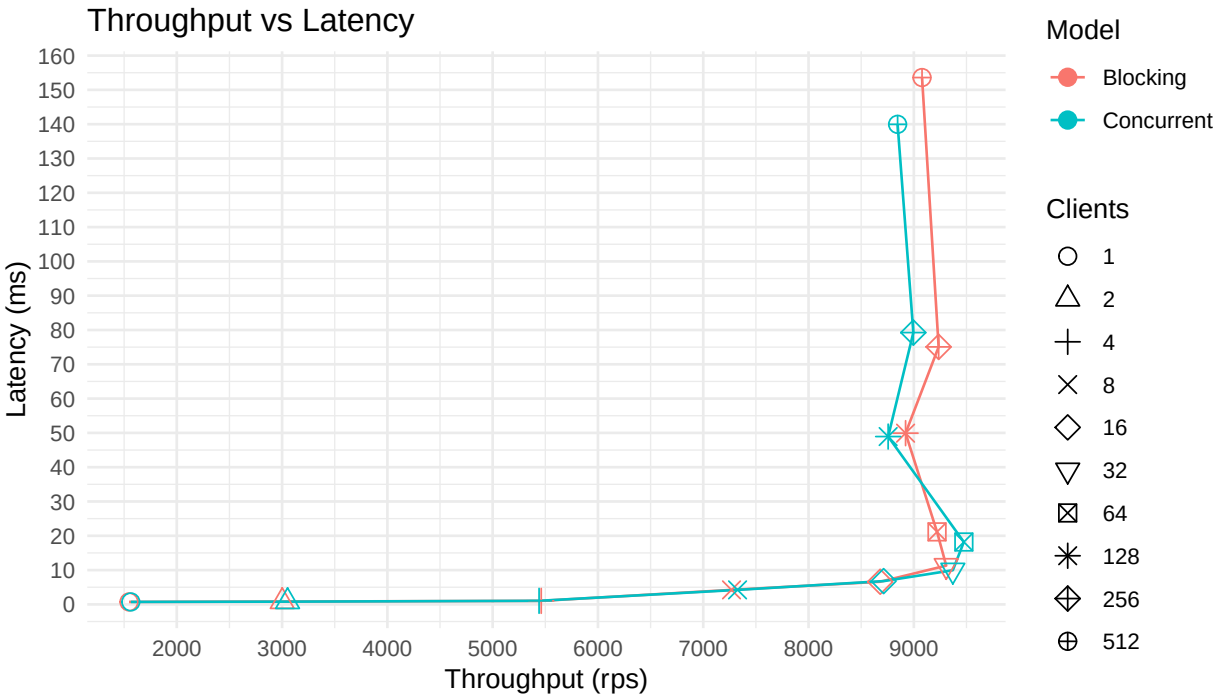
## 2021-12-15 - Ponto de saturação do sistema

| Configuração | Valor | Observação |
|---|---|---|
| Servidores vs Clientes | 1/1 | Node 41-42 |
| Escrita vs Leitura | 100/0 | |
| Chaves | 1500000 | |
| Requests | 5000000 | |
| Threads no cliente | 1-512 | |
| Bytes/Registro | 4kb | |
| Período de Checkpoints | NA | |
| Checkpoints assíncrono | NA | |
| Log assíncrono | - | Log desabilitado |
| Estrutura de dados | java.util.TreeMap e scala.collection.concurrent.TrieMap | |
| Commit | ce0d34d | 275a205 |

Neste experimento buscamos encontrar o ponto de saturação do sistema variando o número de threads em um sistema com checkpoints desligados.


Throughput by number of threads
Throughput (rps)
9500
9000
8500
8000
7500
7000
6500
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1
2
4
8
16
32
64
128
256
512
Threads
Model
Blocking
Concurrent

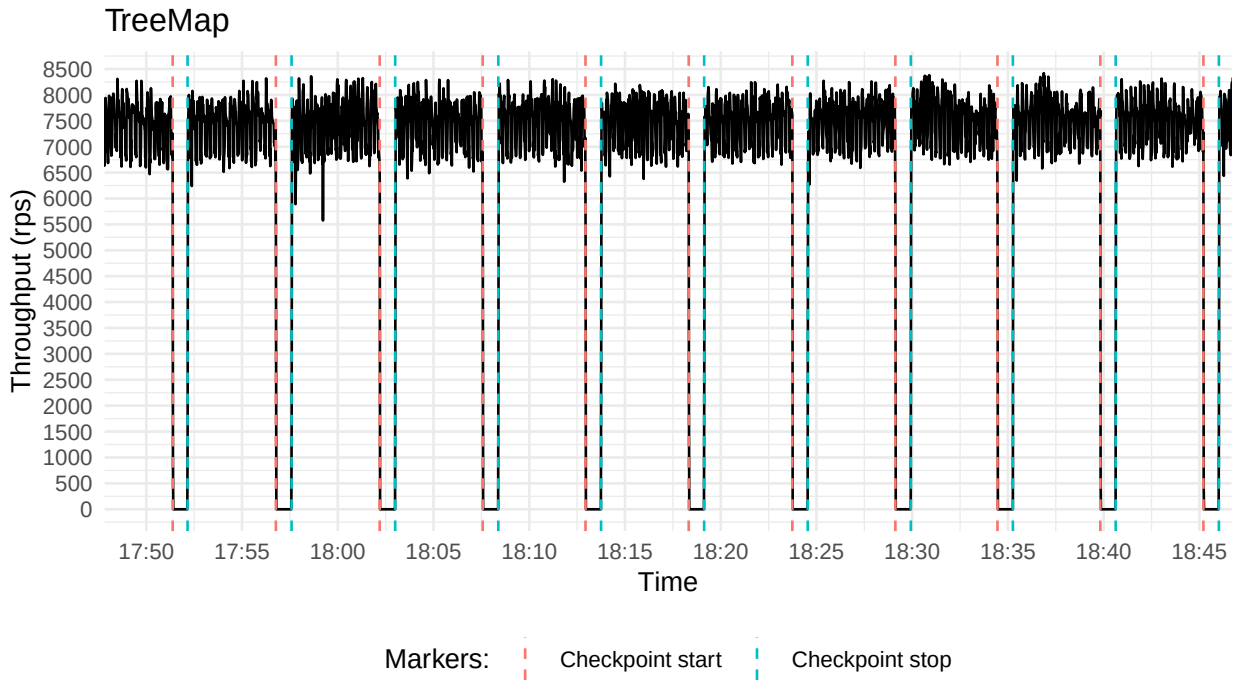

Latency by number of threads
Throughput (rps)
160
150
140
130
120
110
100
90
80
70
60
50
40
30
20
10
0
1
2
4
8
16
32
64
128
256
512
Threads
Model
Blocking
Concurrent

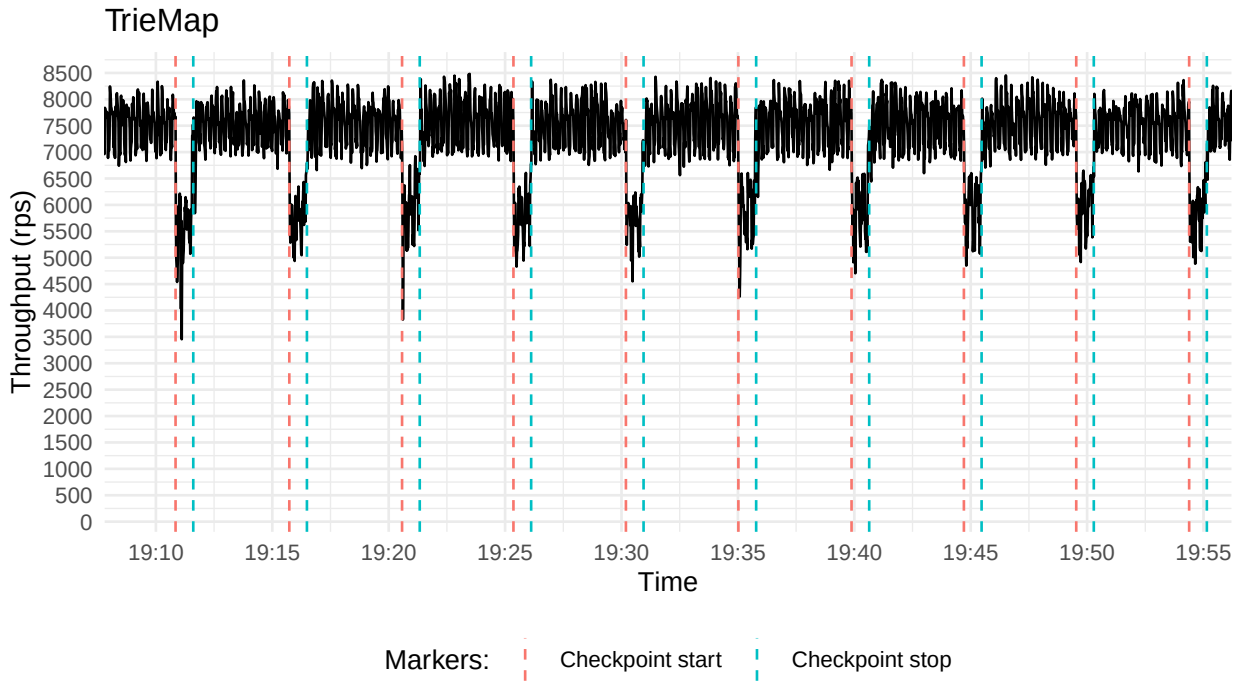



### 2020-02-27 - Experimentos de longa duração em regime

| Configuração | Valor | Observação |
|---|---|---|
| Servidores vs Clientes | 1/1 | Node 41-42 |
| Escrita vs Leitura | 100/0 | |
| Chaves | 1500000 | |
| Requests | 1h | |
| Threads no cliente | 8 | |
| Bytes/Registro | 4kb | |
| Período de Checkpoints | 2000000 | |
| Checkpoints assíncrono | Sim | |
| Log assíncrono | - | Log desabilitado |
| Estrutura de dados | java.util.TreeMap e scala.collection.concurrent.TrieMap | |
| Commit | ce0d34d | |

Re-executamos os experimentos de longa duração em regime (8 threads).




**Vazão média em relação ao modelo de concorrência**

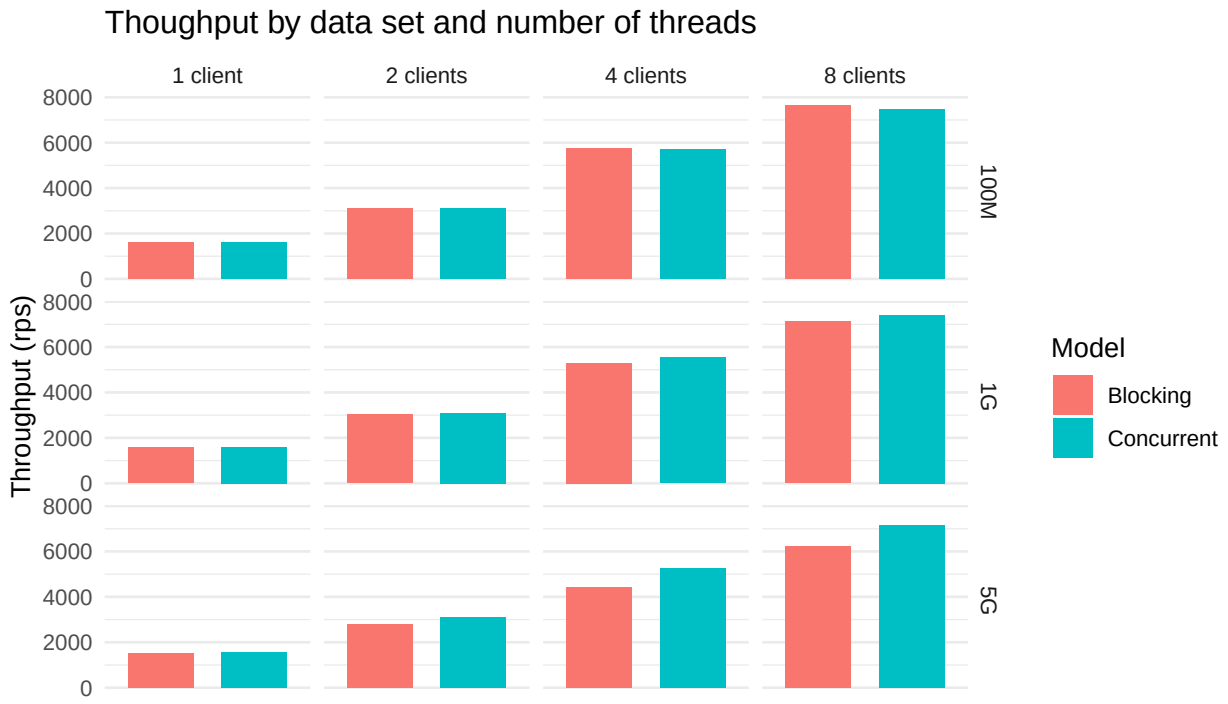
| Model | Mean | SD | SE |
|---|---|---|---|
| Blocking | 6375.02 | 2685.29 | 44.75 |
| Concurrent | 7269.79 | 780.73 | 13.01 |
| - | 14.04 | -70.93 | -70.93 |

**Vazão média fora e dentro de checkpoints**

| Checkpoint | Mean | SD | SE |
|---|---|---|---|
| Out | 7518.87 | 494.47 | 8.96 |
| In | 5894.44 | 625.36 | 26.62 |
| - | -21.60 | 26.47 | 197.18 |

## 2020-02-28 - Matrix de experimentos Threads x Tamanho do estado

| Configuração | Valor | Observação |
|---|---|---|
| Servidores vs Clientes | 1/1 | Node 41-42 |
| Escrita vs Leitura | 100/0 | |
| Chaves | 25600 (100MB), 262144 (1G), 1310720 (5G) | |
| Requests | 15m | |
| Threads no cliente | 1, 2, 4, 8 | |
| Bytes/Registro | 4kb | |
| Período de Checkpoints | 2000000 | |
| Checkpoints assíncrono | Sim | |
| Log assíncrono | - | Log desabilitado |
| Estrutura de dados | java.util.TreeMap e scala.collection.concurrent.TrieMap | |
| Commit | ce0d34d | |



**2020-03-06 - Microbenchmarks das estruturas de dados utilizadas**