COMPARAÇÃO DE DESEMPENHO

***ConcurrentHashMap vs Collections.synchronizedMap*:**

Foram executadas um milhão de chamadas do método get() e um milhão de chamadas do método put(), esses dados foram coletados e analisados separadamente para determinar qual das estruturas apresenta o melhor desempenho para diferentes números de threads, sendo coletado 50 amostras para cada quantidade de threads afim de se obter um valor consistente.

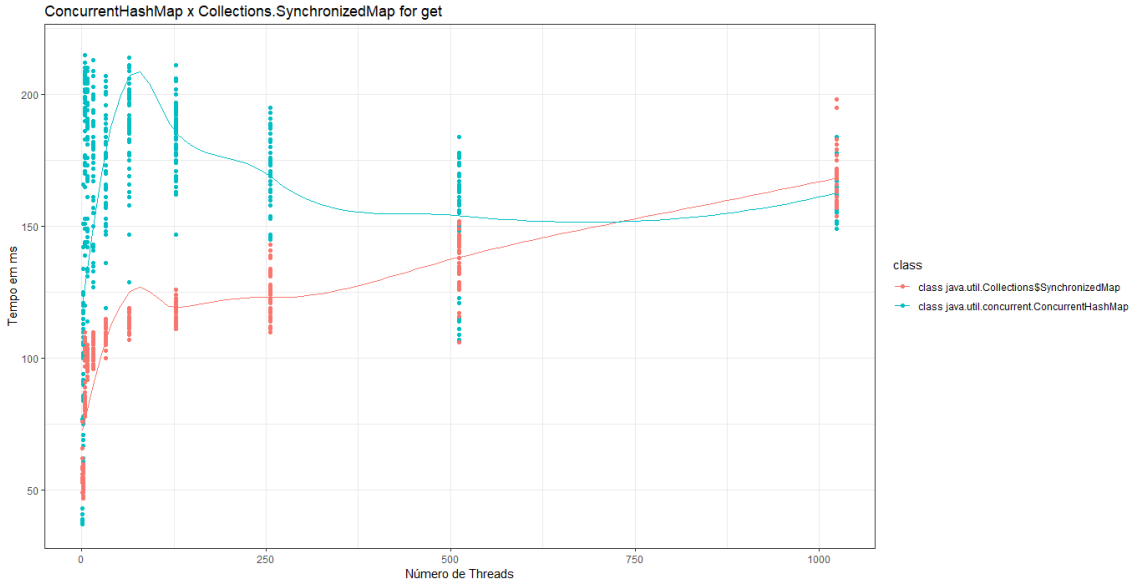

Gráfico 1

O *Gráfico 1*, com base nos pontos distribuídos, apresenta a curva de custo em milissegundos da execução do método *get()* para determinados número de threads. Neste, está evidente que para um número pequeno de threads (menos que 64) a execução do método *get()* na *Collections.SynchronizedMap* é mais rápido, este custo tende a se igualar a partir daí. O *get()* é apenas uma execução de leitura, isso não é sincronizado nas duas estruturas, por isso apresentam um tempo de execução similar para esta função.

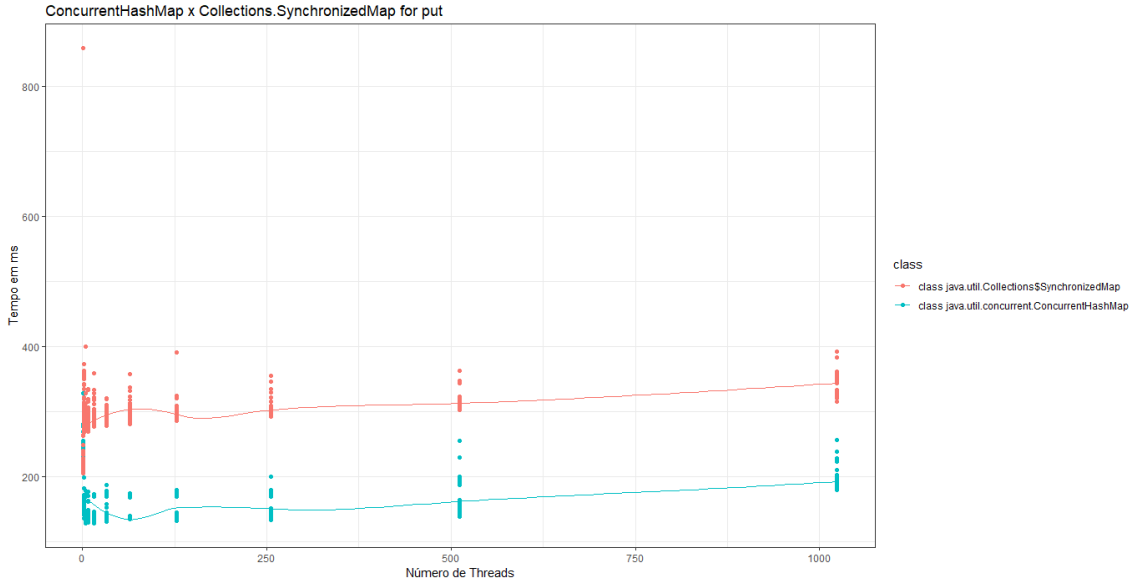

Gráfico 2

O *Gráfico 2* apresenta uma pequena crescente linear no tempo de execução do método *put()* em relação ao crescente número de threads, sendo que, para qualquer número de threads, a estrutura *ConcurrentHashMap* é aproximadamente 200 milissegundos mais rápida. *ConcurrentHashMap* cria múltiplos segmentos de concorrência, por isso, funções executadas para alterar diferentes áreas do código podem ocorrer em paralelo.

***CopyOnWriteArrayList vs Collections.synchronizedList*:**

Foram executadas vinte mil chamadas do método *get()* e vinte mil chamadas do método *add(),* esses dados foram coletados e analisados separadamente para determinar qual das estruturas apresenta o melhor desempenho para diferentes números de threads, sendo coletado 50 amostras para cada quantidade de threads afim de se obter um valor consistente.

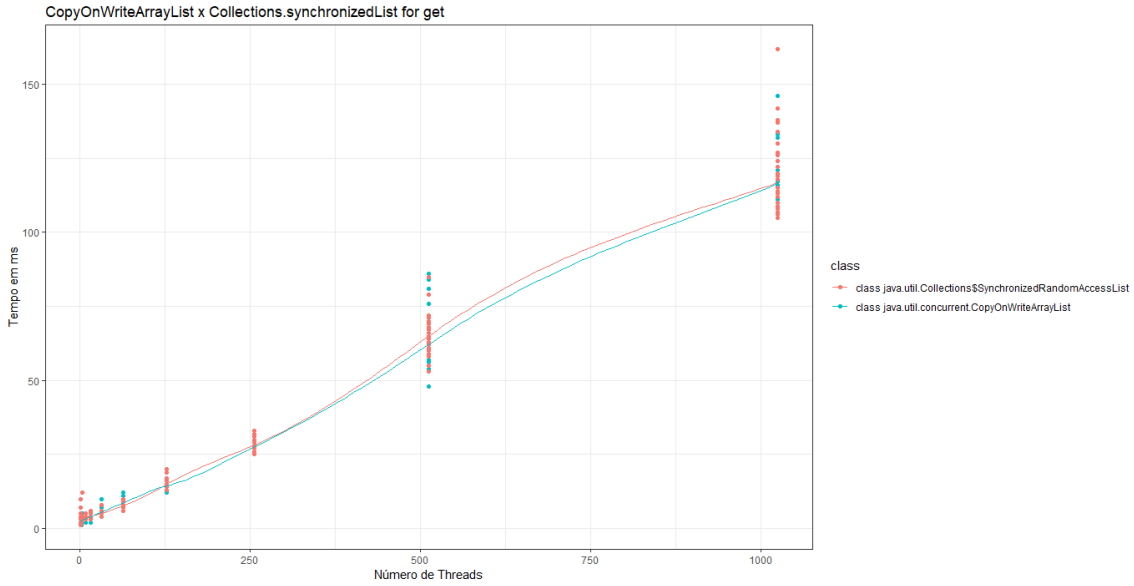

Gráfico 3

O *Gráfico 3* mostra uma reta crescente do tempo de execução em milissegundos em relação ao crescente número de threads. Sendo, para o método *get()*, o mesmo custo de tempo nas duas estruturas. CopyOnWriteArrayList é uma lista otimizada para leitura, tendo um

array imutável internamente. Este array é acessado por métodos não-sincronizados. Múltiplas threads podem acessar estes métodos de maneira segura, uma vez que o array interno é imutável. Por isso o mesmo tempo de execução do *Collections.synchronizedList.*

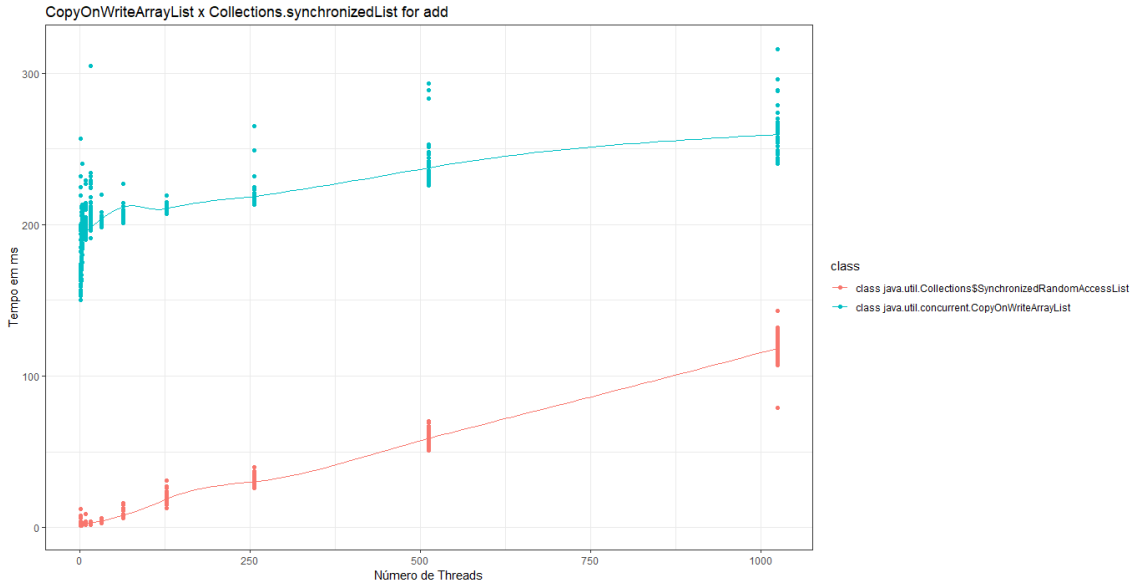

Gráfico 4

O Gráfico 4 apresenta uma reta crescente e linear do tempo de execução em milissegundos em relação ao crescente número de threads. Sendo que, a *CopyOnWriteArrayList* é até 200 milissegundos mais lenta para executar o mesmo número de métodos *add()* para o mesmo número de threads da estrutura *Collections.synchronizedList*. Na estrutura *CopyOnWriteArrayList*, métodos que modificam a lista são sincronizados, o conteúdo da lista antiga é copiado em uma nova sempre que for modificada, por este motivo tem um constante tempo de execução menor que a *Collections.synchronizedList*.