PPD - Relatório 2

Eduardo Montagner de Moraes Sarmento e Pedro Paternostro

Julho de 2021

1 Introdução

O objetivo deste trabalho é a implementação de um sistema cliente/servidor por meio de middleware RPC ou RMI e a observação do comportamento do respectivo programa com diferentes quantidades de processos ou threads.

2 Metodologia

Para a nossa implementação, escolhemos a linguagem Python, utilizando o XML-RPC como middleware e utilizamos processos como mecanismo de paralelismo. A linguagem Python possui tabelas hash de tamanho dinâmico, implementadas com o tipo nativo dicionario, portanto não é necessário criar antecipadamente uma hashtable com o tamanho pré-definido.

O código utilizado para a execução do trabalho, disponível neste mesmo repositório, é composto de três arquivos principais. *Exercicio2_server.py* representa o servidor, implementa e registra funções de get, que retorna numero contido pela tabela hash dado a chave caso ele exista, e retorna NONE caso ele não exista, e post, que dado uma chave e um numero insere este numero na tabela hash na posição apontada pela chave, referentes a sua tabela hash e abre a porta 8000 para requisições, *Exercicio2_client.py* representa o cliente, conecta com o servidor e gera a quantidade especificada de números inteiros aleatorios, fazendo chamadas de post, utilizando o numero aleatorio criado tanto para a chave quanto para o numero a ser inserido, e get, que retorna o numero que acabou de ser inserido, enquanto *Runner.py* gerencia a criação e execução de processos clientes (*Exercicio2_client.py*) em paralelo.

Para a execução do programa, basta executar em dois terminais diferentes o arquivo *Exercicio2_server.py* sem argumentos e o arquivo *Runner.py* com o número de threads e o número de inserções respectivamente como argumento (e.g. *python Runner.py 4 800*). Em nossa execução um computador com as seguintes configurações foi utilizado:

Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz, 3401 Mhz, 4 Cores, 8 Logical Processors / 8 GB RAM DDR3

3 Resultados

Processos	Requisições	Tempo(s)
2	30000	22538.26
4	30000	11414.09
8	30000	5877.85

Tabela 1: Resultados de execução

Analisando a Tabela 1, nota-se que a o número de processos paralelos faz grande diferença no tempo total requerido para processar as requisições. De modo que processar todas as requisições com 8 processos paralelos demora aproximadamente $\frac{1}{4}$ do tempo necessário para processar a mesma quantidade de requisições com dois processos paralelos.

Não foi necessario implementar maneiras de se controlar a concorrencia pois a blibioteca utilizada do Python (XML-RPC) ja faz isso, o servidor enfilera as requisições e as executa na ordem em que elas são feitas. A implementação de hashtable do Python (dictionary) também trata colisões que eventualmente possam acontecer.