Nome: Seong Eun Kim
RA: 177143

Teste 02

Introdução

Nesse teste, foi implementado em Python o algoritmo de disseminação de informação entre processos por gossiping utilizando sockets TCP/IP. Na implementação, foi utilizado o ZeroMQ como sistema de troca de mensagens.

Algoritmo

Primeiramente, foi criada uma rede de processos que se conectam via sockets, a qual possui uma array de structures compartilhada entre seus nós para armazenar informações experimentais sobre cada processo. Em cada nó, rodam duas threads: uma do cliente, responsável em transmitir a informação; e outra do servidor, responsável por ouvir a informação. Foi usado o multithreading para que o cliente e o servidor pudessem funcionar sincronizadamente, pois as funções de comunicação - send() e recv() - bloqueiam a thread até retornarem.
O cliente possui um loop de envio, no qual se ele possuir a informação, ele escolhe uma porta randomicamente para se conectar e enviar a mensagem. Se o processo recipiente já possuía a informação, o remetente para o processo de disseminação com probabilidade 1/k. Caso contrário, ele continua enviando mensagens para portas aleatórias. O servidor possui um loop que aguarda uma mensagem e faz o tratamento adequado para quando o nó já possuía ou não a informação. Ambos os loops possuíam um timeout após o início do processo de disseminação (após o último processo ser criado), dando tempo suficiente para que a maioria dos processos tenham recebido a mensagem e para que os servidores não fiquem esperando mensagens para sempre, depois que todos os remetentes tenham sido desativados. Ao final, os sockets são fechados, e é dado 100ms para que as mensagens ainda na memória sejam enviadas. Então, o contexto é fechado.

Testes

Ao executar o algoritmo em OSx em um MacBook Pro com processador 2,7GHz Intel Core i5, pode-se rodar até 508 processos. Para números maiores do que esse, ocorre um erro do tipo "Too many files open", possivelmente por causa do limite de arquivos do sistema operacional. Como com um timeout de mais de 20 segundos, quase todos os processos ficam sabendo da informação para variados valores de *k*, ajustou-se o timeout para 12 segundos. Dessa forma,

pode-se fazer uma análise dos dados coletados conforme *k* varia.
Resultados obtidos testando com 400 processos:

k	2	4	6	8
Mínimo de tentativas	0	0	0	0
Máximo de tentativas	6	7	6	6
Média de tentativas	2.410	3.042	2.998	3.312
Parcela de sucesso	38%	32%	32%	29%
Processos com informação	368	387	380	389
Tempo	22s	22s	22s	22s

Quando *k=5* e variando-se o número total de processos, obtivemos:

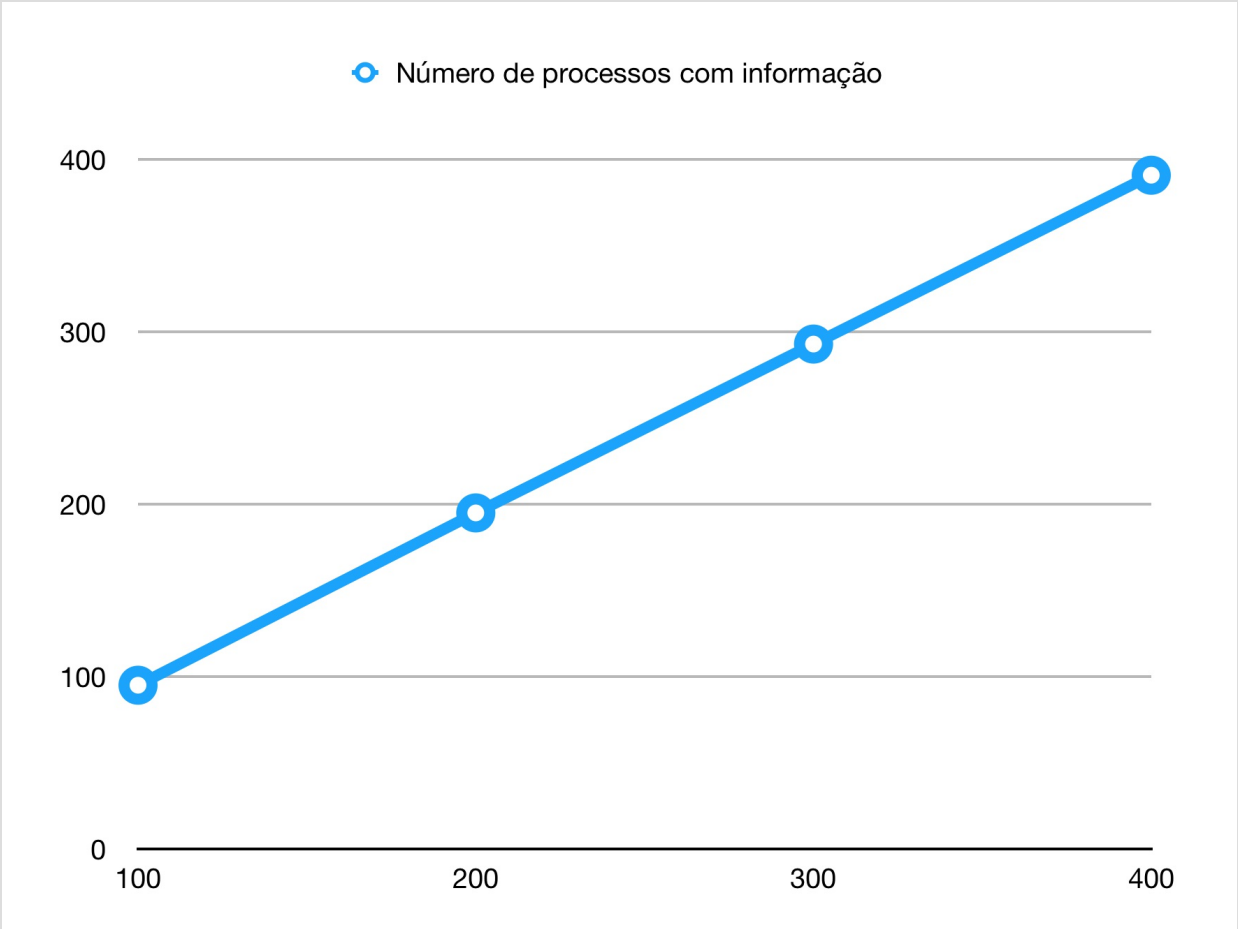

Gráfico do número de processos com a informação em relação ao número total de processos

Análise

Com os valores obtidos, podemos observar que conforme *k* cresce, aumenta-se o número de processos com a informação, pois diminui-se a probabilidade deles morrerem. Assim, o número médio de tentativas para distribuir a mensagem aumenta, enquanto a parcela de sucesso da disseminação diminui. Apesar da maior chance de fracasso ao transmitir a mensagem, por haver mais processos remetentes com *k* maiores, mais processos têm a informação no final.
Pelo gráfico, observa-se também que a dispersão de mensagens se dá de forma linear conforme o número de processos aumenta. Isso se deve porque, inicialmente, há maior chance de sucesso na disseminação, pois há vários processos sem informação disponíveis, mas, ao longo do tempo, há maior chance de fracasso.

Discussão e conclusão

Na elaboração do algoritmo, os principais problemas e/ou dificuldades foram na implementação do ZMQ devido à sua documentação, problemas com o limite de arquivos abertos do sistema operacional e trabalhar com centenas de multiprocessos, o que por vezes causava resultados inconsistentes. Acredita-se que uma provável causa para os problemas envolvendo multiprocessos seria a escolha do sistema operacional que foi usado no desenvolvimento desse projeto, pois o OSx apresenta problemas com a criação de muitos arquivos, mesmo tentando usar comandos que alteram o limite do sistema.
Nesse trabalho, pode-se estudar a implementação de algoritmos de disseminação, o funcionamento de sockets e o uso de ZMQ como sistema de trocas de mensagens. Por fim, de forma geral, pode-se dizer que para os testes realizados, os resultados foram consistentes com o esperado.