Sudoku Distributed System

Computação Distribuída

Turma P1

Docente: Diogo Gomes

Discentes:

Shelton Lázio Agostinho – 115697 – sheltonagostinho@ua.pt

Gabriel Santos – 113682 – gvds@ua.pt

8 de Junho de 2024

Introdução

O objetivo deste trabalho era realizar um sistema distribuído capaz de resolver vários puzzles sudoku. Neste projeto implementámos métodos realizados noutros trabalhos na mesma disciplina, mais precisamente os guiões práticos que nos foram fornecidos ao longo do ano, como por exemplo:

- Comunicação TCP por sockets;
- A comunicação Peer to Peer, no nosso caso todos comunicam com todos na rede;

Excluindo o load balancing devido à forma de como distribuímos as tarefas.

Foi-nos também proposto o desenvolvimento de um servidor HTTP, e consequentemente também nos foram disponibilizados links onde poderíamos aprender a lidar com esse tipo de sistema. O trabalho consistia em enviar um ou mais problemas de sudoku para um node, e a partir deste distribuir soluções por diversos outros nodes, com o objetivo de minimizar o tempo de resolução deste puzzle. A solução do grupo deveria cumprir uma lista de requisitos:

- Desenvolver um sistema distribuído peer to peer;
- Criar uma API web HTTP (com endpoints /stats; /network; /solve);
- Escolha de um protocolo de comunicação;
- Todos os nós devem aceitar pedidos através da interface HTTP local;
- Encontrar solução no menor tempo possível (Mais nós, é expectável solução mais rápida);
- Solução tolerante a falhas;
- P2P deve ser dinâmica;
- A solução deve funcionar em 2 dois computadores em rede;
- Não se deve usar bibliotecas externas;
- Deve poder ser controlada a velocidade à qual o sistema resolve os problemas (nó deve poder ser penalizado com atraso);
- Cada nó deve ser executado na linha de comandos;
- Deve-se poder receber mais que um puzzle em simultâneo.

Algoritmo

O algoritmo de sudoku, como nos foi indicado, é uma solução através de "brute force" que consiste na geração de várias soluções possível e apenas posteriormente à sua verificação pelo código que nos foi fornecido, na função check, pelo professor.

O nosso algoritmo de sudoku começa por fazer todas as permutações possíveis de cada linha, de forma isolada. Isto irá permitir com que distribuamos a resolução, que será feita pela atribuição de um conjunto de possíveis soluções de 2 ou 3 linhas que no final deverá retornar as combinações que fazem sentido.

Dentro da classe Peer onde temos o protocolo de comunicação também temos obviamente que conter a função solve_sudoku(). Primeiramente usa a função

get_possible_solutions_for_row(), para as 9 linhas tendo assim todos os valores possíveis do puzzle. Depois, passa pela segunda faze que é a primeira parte distribuída, em que enviamos linhas do puzzle, duas a duas, para analise a um peer disponível, mas se não tiver um peer disponível o próprio nó resolve.

Depois dessa fase, é feita a distribuição das linhas resultantes da fase anterior, desta vez 3 a 3, seguindo a logica: enviar a nós disponíveis e se não tiver, ele mesmo resolve.

Por fim, o nó que recebe o request junta todas as combinações (3 a 3) e avalia qual combinação é que é a solução, imprimindo no caso positivo e retornando.

Não fizemos o load balance por conta do número e forma com que as atividades são realizadas. No entanto existia a possibilidade de distribuir mais as atividades.

Explicação sumária de algumas funções:

- oferecer_tarefa_especifica(): Depois que um nó que resolvia uma tarefa se desconecta, é usada para recuperar a tarefa e oferecer a um peer disponível ou para o próprio nó.
- dividr_tarefas(): Faz gestão da distribuição das tarefas entre os nós da rede. Verifica se há pares disponíveis, senão executa localmente a tarefa.
- my_work(): Esta função executa o trabalho atribuído a este próprio nó.
- validar_2_combinacoes(): Valida as combinações dois a dois.
- combinar_2_linhas(): usa a função acima para retornar a opções validas de combinação de linhas 2 a 2.
- validar_3_combinacoes(): Valida uma combinação de 3 linhas.
- combinar_3_linhas: Igual à função combinar 2 linhas, mas com 3 linhas.

Comunicação

Para a comunicação os nós comunicam-se com todos os outros. O primeiro nó não comunica com minguem no início, e a cada nó que se junta a rede recebe o conjunto de todos outros nos d rede para que possa se conectar. Para cada nó existe uma função que envia keep alives em intervalos de 1 segundo. A deteção de no desconectado é feito 3 segundo depois. Temos uma estrutura de dados para a gestão da rede e das resoluções.

Mensagens trocadas:
Mensagem Simples (message):

- Quando a mensagem recebida tem o comando "message", ela simplesmente imprime o conteúdo da mensagem.

Informações dos Peers (peers_info):

- Atualiza o número de validações e o número de problemas resolvidos.
- lista de novos peers que ainda não conhece.
- Envia uma mensagem de reconhecimento (peers_ack) de volta para o peer que enviou a informação, contendo a lista atualizada de peers e informações do próprio socket.

Resposta do Peer (peers_ack):

- Atualiza a lista de peers conectados com as informações do peer que enviou o resposta.
- Conecta-se a novos peers listados na mensagem de reconhecimento, se existissem.

Envio de keep alives (ack):

Atualiza o tempo de última comunicação, informações de rede e validações para um peer específico.

Problema Resolvido (solved):

- Incrementa a contagem de problemas resolvidos.

Atualização de Validações (validations):

- Atualiza o número de validações totais com os dados recebidos.

Tarefa (task):

- Envio de tarefas
- Se a tarefa envolve duas linhas, combina essas duas linhas e prepara uma resposta.
- Se a tarefa envolve três linhas, combina essas três linhas e prepara uma resposta.
- Envia a resposta da tarefa para o peer que a solicitou.

Resposta de Tarefa (task_resp):

- Atualiza a lista de validações com os dados recebidos.
- Remove a responsabilidade do peer da lista de tarefas pendentes.
- Imprime as combinações recebidas do peer.
- Atualiza as possíveis soluções com as combinações recebidas.
- Marca o peer como disponível novamente para novas tarefas.

Configuração dos Nós

Cada nó na rede é um servidor HTTP que se comunica com outros nós usando um protocolo de comunicação peer-to-peer, onde estes eles estabelecem uma conexão TCP entre si para trocar mensagens. Esta conexão é realizada através de sockets. Quando o cliente deseja informação da rede ou introduzir um puzzle, ele usa curl para realizar a comunicação HTTP. A configuração inicial dos nós inclui a definição de portas para comunicação HTTP e P2P. Abaixo estão exemplos de comandos utilizados para iniciar um nó:

Primeiro nó: python3 node.py -s 7001 -p 8001 –h 1

Segundo nó: python3 node.py -s 7002 -p 8002 –h 1 -a 198.89.123.54:7001

Comunicação

Os nós comunicam-se utilizando sockets numa ligação TCP para trocar informações e coordenar a resolução dos puzzles de Sudoku. A comunicação é estabelecida através de “endereço:porta”e

específicos que cada nó expõe. As mensagens são trocadas em JSON. Os comandos suportados e assinalados nos diferentes campos das mensagens são: ("message", "peers_info", "peers_ack", "ack", "solved", "task"e "task_resp"). A sincronização e verificação de peers é realizada na função alive(), onde eles enviam mensagens "ack" enquanto o nó se mantiver vivo. Temos também uma função "are_peers_alive()"que verifica se houve algum nó que se desconectou.

Tempo relativo ao número de nós - Gráfico

Características da solução:

- Solução testada a mandar curl para o mesmo node / e diferentes, não divergia
- Testes com Sudoku function: base_delay = 0, interval = 1, threshold = 1, handicap = 1 (default)
- Ancoragem dos nodes seguida / ao mesmo

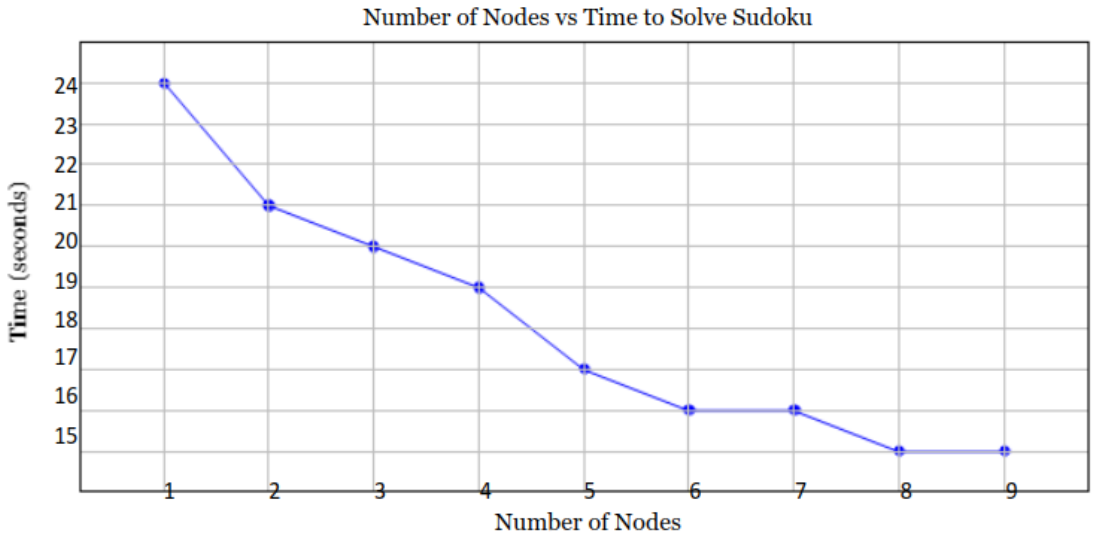

Figura 5.1: Comparison of Number of Nodes Used and Time Spent Solving Sudoku