deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Automotive App

Projeto em Engenharia Informática

Filipe Miguel Neto Viseu - 119192
Samuel Marques Vinhas - 119405
Eduardo Galveia Romano - 118736
D - 1
D - 2

Ano letivo 2025/2026

Conteúdo

1. Introdução

Este projeto implementa um sistema distribuído peer-to-peer para avaliação automática de código Python (através do pytest). O sistema permite distribuir a execução de testes entre múltiplos nós, garantindo mínima tolerância a falhas e balanceamento de carga. O sistema que implementámos é composto por:

- **Nós P2P**: Cada instância do sistema que pode atuar como supervisor ou worker, cada nó mantém as suas próprias estatísticas e o sistema agrega automaticamente estatísticas de todos os nós
- **API REST**: Interface para submissão e monitorização de avaliações, com diversos endpoints listados abaixo
- **Keepalive System**: Monitorização contínua do estado dos nós e deteção automática de nós "mortos"
- **Load Balancer**: Distribuição inteligente e equilibrada de tarefas
- **Descoberta de nós**: Descoberta automática de novos nós na rede

2. Arquitetura

Para este projeto nós pensámos na seguinte arquitetura:

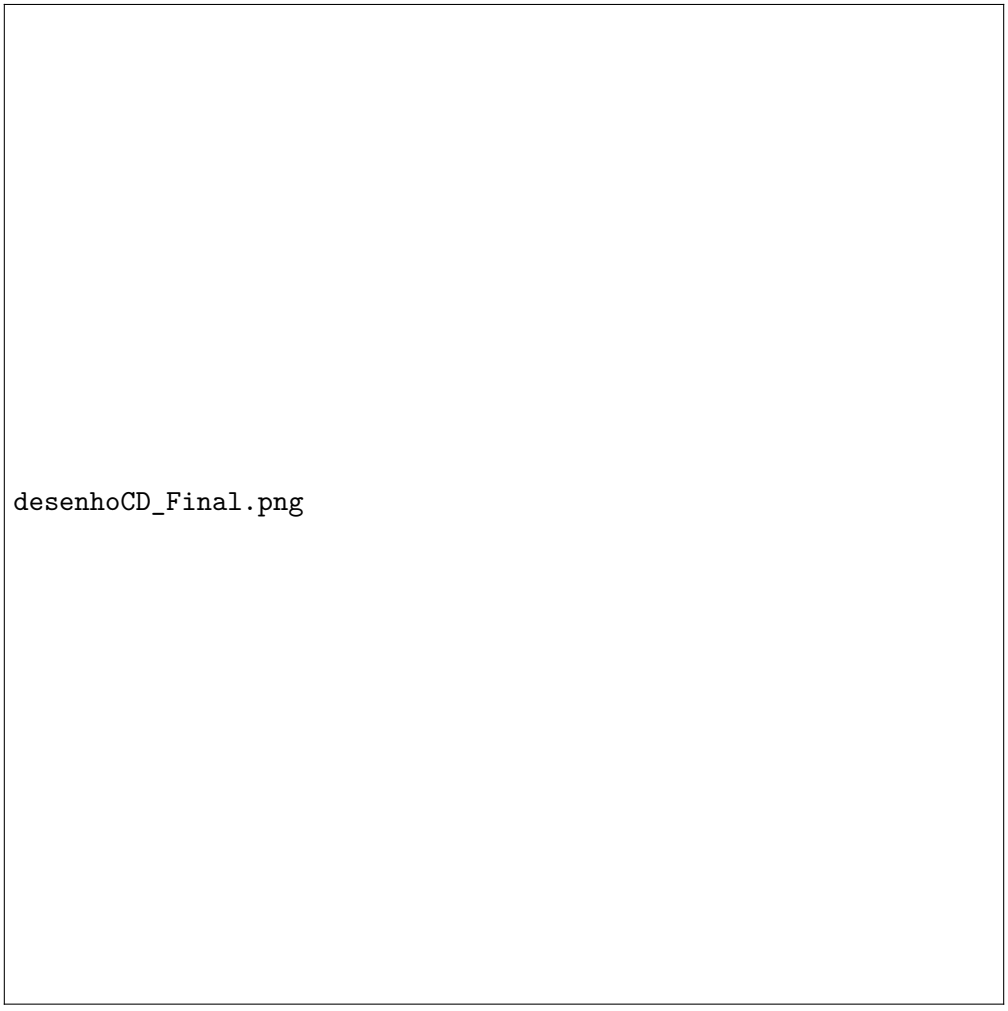

Figura 2.1: Ideia Geral

Qualquer um dos nós pode receber uma avaliação. Assim que a recebe este torna-se o "supervisor"da avaliação, extrai e manda um /execute_module aos outros nós. Ao receber esta mensagem os nós mandam um /getEvaluationContent para saber o que pre-

cisam de avaliar. O supervisor distribui então aquilo que extraiu e espera por resultados (post de /evaluation_result). Enquanto espera, o supervisor continua a trabalhar, executando os seus "pytest". Depois de ter todos os resultados e de perceber que a avaliação foi concluída o supervisor manda um /evaluation_done para os outros nós saberem que aquela avaliação acabou e apaga os ficheiros temporários.

3. Endpoints

Endpoint	Método	Descrição
`/stats`	GET	Retorna estatísticas globais da rede incluindo todos os nós
`/network`	GET	Retorna a topologia atual da rede P2P
`/evaluation`	GET	Lista todas as avaliações na rede
`/evaluation/<id>`	GET	Status detalhado de uma avaliação específica
`/evaluation`	POST	Submete uma nova avaliação (ZIP ou repositórios Git)
`/join`	POST	Permite que um novo nó se junte à rede
`/mystats`	POST	Heartbeat para troca de estatísticas entre nós
`/execute_module`	POST	Executa um módulo de teste específico
`/get_evaluation_content`	POST	Obtém conteúdo de uma avaliação
`/evaluation_result`	POST	Recebe resultados de execução de testes
`/evaluation_done`	POST	Notifica conclusão da avaliação e limpa a pasta da mesma

4. Load Balancer

Para a realização deste projeto optámos por reaproveitar algo que tinhamos feito em aula e usar o algoritmo de Least Connections para distribuição equilibrada entre os nós disponíveis.

```
class LeastConnections:
    def __init__(self, servers):
        self.servers = servers
        self.evaluations = {server: 0 for server in servers}

    def add_server(self, server):
        if server not in self.evaluations:
            self.evaluations[server] = 0
            self.servers.append(server)

    def remove_server(self, server):
        if server in self.evaluations:
            del self.evaluations[server]
            self.servers.remove(server)

    def select_server(self):
        server = min(self.evaluations, key=self.evaluations.get)
        self.evaluations[server] += 1
        if not isinstance(server, str):
            server = server.address
        return server

    def update(self, server, active_evaluations):
        if server in self.evaluations:
            self.evaluations[server] = active_evaluations
        else:
            self.add_server(server)
            self.evaluations[server] = active_evaluations
        pass
```

Código 4.1: LeastConnections

5. Tolerância a falhas

O sistema implementa algumas estratégias de tolerância a falhas:

5.1 Keepalive

Cada nó monitoriza continuamente o estado dos outros nós através de mensagens periódicas (/mystats):

```
def stats_request(self):
    while True:
        time.sleep(self.heartbeat_interval)
        for node in list(self.network.keys()):
            if node != self.address:
                try:
                    res = requests.post(
                        f"http://{node}/mystats",
                        json={
                            "sender": self.address
                            #"stats": self.stats,
                            #"network": self.peer_list,
                            #"active_evaluations": self.active_evaluations,
                            #"evaluations": self.evaluations
                        },
                        timeout=3
                    )
                    if res.status_code == 200:
                        self.handle_stats_reply(res.json())
                    else:
                        print(f"Node {node} returned status {res.status_code}")
                        self.handle_node_failure(node)
                except (requests.RequestException, requests.Timeout) as e:
                    print(f"Error connecting to {node}: {e}")
                    self.handle_node_failure(node)
```

Código 5.1: KeepAlive

5.2 Deteção de nó inativo

Quando um nó falha (deixa de mandar stats) os outros nós detetam, removem automaticamente da sua rede e as tarefas são redistribuidas :

```
def handle_node_failure(self, node):
    if node in self.network:
        print(f"Removing inactive node: {node}")
        del self.network[node]
        if node in self.peer_list:
            self.peer_list.remove(node)
        self.load_balancer.remove_server(node)

        # Redistribute pending modules
        for eval_id, eval in self.evaluations.items():
            for module in eval.get("modules_pending", []):
                target = self._select_target_node()
                self._send_module_to_node(target, eval_id, module)

        self.update_all_stats()
```

Código 5.2: Deteção