Projecto Computação Distribuída

Licenciatura em Engenharia Informática - Computação Distribuída

Distributed CD Tester

Revisão:
10 de Abril 2025
Docentes:
- Diogo Gomes (dgomes@ua.pt)
- Nuno Lau (nunolau@ua.pt)
- Alfredo Matos (alfredo.matos@ua.pt)

Prazo:
5 de Junho – 00h00

Conceitos a abordar:

- Sockets
- Marshalling
- Fault Tolerance
- Distributed workloads

Introdução

Na disciplina de Computação Distribuída já contamos com testes aos mini-projectos. Estes testes podem ser executados localmente ou através de GitHub actions que executam na cloud os testes e disponibilizam resultados individuais.

No âmbito deste projecto pretende-se desenvolver um sistema distribuído recorrendo a vários nós computacionais, que executam dentro de Docker Containers, para acelerar o processo de teste de todos os projectos e calcular uma nota final, que será a média das notas de cada projecto.

O objetivo deste projeto é, pois desenvolver um sistema distribuído que automatize os testes de projetos Python de Computação Distribuída alojados no GitHub. O sistema deve ser capaz de executar conjuntos de testes `pytest` de forma distribuída com distribuição e agregação dos resultados de testes de forma eficiente.

Os alunos irão desenvolver um sistema peer-to-peer (P2P) onde vários nós contribuem para a execução de casos de teste em paralelo. O sistema deve ser resiliente a falhas de nós e permitir a adição e remoção dinâmica de nós.

Protocolo de funcionamento:

Cada nó do sistema distribuído irá:

1. Receber trabalho através de 2 métodos:
    a. Upload de um ficheiro ZIP que contém um ou mais projectos e respectivos testes.
    b. Envio de uma lista de ligações (URLs) correspondentes aos diversos repositórios no GitHub e respectivo Token para autorização.
2. Extrair e distribuir os módulos/ficheiros de teste entre os nós disponíveis:
    a. atribuir aos nós disponíveis os módulos/ficheiros de teste.
    b. agregar os resultados e calcular o relatório final com nota por projecto (% de testes que passam) e global (média de todos projectos).
3. Devolver os resultados dos testes através de uma API web.

Para interagir com o seu sistema distribuído deve criar uma API web (HTTP) que recebe o pedido de trabalho definido em 1) através dos endpoints `/upload` e `/evaluation`, monitorizar o estado do sistema como definido em 2) através de `/evaluation/<id>` e permite a consulta dos resultados finais 3) também em `/evaluation/<id>`. Estes endpoints devem ser definidos pelo grupo e documentados em **protocolo.pdf** a colocar na raiz do repositório.
Todos os nós do sistema distribuído devem poder aceitar pedidos de avaliação/progresso de projectos através da sua interface HTTP.

Paralelamente e para fins de avaliação, deverá implementar o endpoint `/stats` onde seja possível consultar o progresso da tarefa desde a submissão até obter o resultado final (persistente). E ainda o endpoint /network onde seja possível monitorizar a rede P2P através dos endereços dos nós e ligações entre os mesmos.
Estes endpoints estão exemplificados abaixo, e deve seguir o formato JSON do exemplo à risca.

`/stats`

```
{
    "all": {
        "failed": 2,
        "passed": 30,
        "projects": 2,
        "evaluations": 3
    },
```

```
    "nodes": [
        {
            "address": "192.168.1.100:7000",
            "failed": 2,
            "passed": 10,
            "projects": 1,
            "modules": 1,
            "evaluations": [12345]
        },
        {
            "address": "192.168.1.100:7001",
            "failed": 0,
            "passed": 20,
            "projects": 2,
            "modules": 1,
            "evaluations": [12345, 67890]
        }
    ]
}
```

“all” refere-se a toda a rede e contém estatísticas globais do número de pytest’s que falharam (“failed”) e que passaram (“passed”), este número é correspondente aos testes individuais de cada módulo. Esta estatística inclui informação desde início do sistema e é independente dos nós actualmente ligados.
“Nodes” apresenta uma lista de todos os nós ligados **actualmente** com informação sobre o respectivo endereço e o número de testes que falharam e passaram nesse mesmo nó.

/network

```
{
    "192.168.1.100:7000": [
        "192.168.1.100:7001",
        "192.168.1.100:7002"
    ],
    "192.168.1.100:7001": [
        "192.168.1.100:7000"
    ],
    "192.168.1.100:7002": [
        "192.168.1.100:7000",
        "192.168.1.11:7003"
```

```
    ],
    "192.168.1.11:7003": [
        "192.168.1.100:7002"
    ]
}
```

Apresenta uma lista de todos os nós da rede, para cada nó apresenta a lista dos nós com os quais comunica.

Objetivos

- Implementar um sistema distribuído de arquitetura peer-to-peer.
- Os nós devem comunicar entre si através de um protocolo a ser proposto por cada grupo e com os seus clientes através de uma interface HTTP.
- Todos os nós devem aceitar o ZIP ou lista de URL's com os projectos a testar através da sua interface HTTP local.
- Testar os projectos de CD de um aluno no menor tempo possível (implementação mais rápidas/eficientes corresponderão a melhores notas). O tempo será medido de forma relativa sendo expectável que um maior número de nós implique um tempo de processamento inferior.
- O protocolo P2P deverá ser definido por cada grupo e documentado num ficheiro PDF colocado na raiz do repositório de código e com o nome **protocolo.pdf**.
- Para notas finais mais elevadas (>16), a solução deverá ser tolerante a falhas dos nós sem que existam perdas de informação.
- A rede P2P deve ser dinâmica, isto é, permitir a entrada e saída de nós a qualquer momento (mesmo durante a execução do pytest).

Requisitos da implementação

- A solução deve funcionar em rede com múltiplos containers Docker no próprio computador e containers num 2º computador (para efeitos de avaliação/demonstração 2 computadores devem ser utilizados e demonstrado que ambos computadores estão a trabalhar o mesmo problema).
- O número de processos que contribui para a resolução do problema deve ser arbitrário (não depender de configurações iniciais), deverá ser possível lançar um número arbitrário de containers.
- Não devem recorrer a bibliotecas externas, apenas as bibliotecas standard do python, Flask e Python Request são autorizadas.
- A interface HTTP deve implementar os endpoints:

Endpoint	Método	Descrição

/evaluation	POST	Recebe um ficheiro ZIP que contém 1 ou mais projectos (cada projecto numa pasta diferente).
		Este método deverá devolver um ID.
		
		Deverá ser possivel testar com:
		`curl -X POST http://localhost/upload \`
		`-H "Content-Type: multipart/form-data" \`
		`-F "file=@all_projects.zip"`

/evaluation	POST	Recebe lista de repositórios e auth token.
		Este método deverá devolver um ID.
		
		Deverá ser possivel testar com:
		`curl -X POST http://localhost/evaluation \`
		`-H "Content-Type: application/json" \`
		`-d '{`
		`"auth_token": "123e4567-e89b-12d3-a456-426614174000",`
		`"projects": [ "http://github.com/detiuaveiro/cd_chat_server_1234",`
		`"http://github.com/detiuaveiro/cd_chord_1234" ]`
		`}'`

/evaluation	GET	Lista todos os trabalhos submetidos Retorna no **minimo**: - Id's de todas avaliações
/evaluation/<id>	GET	Permite consultar o estado de execução de uma avaliação Retorna: - % de testes que passam - % de testes que falham - % de testes que passam por projecto - % de testes que falham por projecto - % de testes que passam por módulo

		- % de testes que falham por módulo - # de teste executados, # em execução, # por executar - Nota final por projecto em escala 0..20 - Tempo em segundos para fazer a avaliação
/stats	GET	Informação estatística para efeitos de avaliação (ver exemplo acima)
/network	GET	Disponibiliza informação sobre os nós da rede P2P a que se encontra ligado, isto é, tem conhecimento do seu endereço. (ver exemplo acima)

O endpoint **/evaluation** permite pois receber tanto um ZIP como lista em formato JSON com a avaliação a realizar utilizando ambos o método POST, mas distinguem-se pelo Content-type.

Avaliação

A avaliação deste trabalho será feita através da submissão de código na plataforma GitHub classroom e de um relatório em formato PDF com não mais de 5 páginas colocado no mesmo repositório junto com o código e com o nome relatorio.pdf.

Está em avaliação o protocolo definido e documentado, assim como as *features* implementadas de acordo com os objetivos:

- Serviço web com API REST seguindo os requisitos do trabalho
- Solução distribuída para avaliar todos projectos no mais curto intervalo de tempo utilizando mais do que um nó
- Eficiência da solução desenvolvida (pretende-se um sistema capaz de responder de forma mais rápida possível aos pedidos dos clientes finais)
- Atender a múltiplos clientes em simultâneo, leia-se capacidade de resolver várias avaliações em paralelo
- Atender à possibilidade de ocorrência de falhas, leia-se que qualquer processo/computador pode ser interrompido a qualquer momento.
- Qualidade de Código (padrões, estruturas de dados, documentação)

GitHub Classroom

- Este projeto é realizado em **grupos de 2** alunos.

- Para resolver este projeto deverá começar por aceitar o mesmo em https://classroom.github.com/a/o1T9cPjJ
- Ao aceitar o projeto será criado um repositório online a partir do qual deve fazer um clone local (no seu computador).
- Deverá enviar as suas alterações periodicamente para o repositório e manter-se atento ao canal *#cd* em https://detiuaveiro.slack.com

Notas

Referências

1. https://datatracker.ietf.org/doc/html/rfc2616
2. https://docs.python.org/3/library/http.server.html
3. https://flask.palletsprojects.com/en/stable/
4. https://requests.readthedocs.io/en/latest/
5. https://docs.docker.com/desktop/
6. https://docs.github.com/en/rest
7. https://docs.github.com/en/rest/authentication/authenticating-to-the-rest-api
8. https://docs.github.com/en/rest/repos/repos#list-repositories-for-a-user