

SISTEMAS DISTRIBUÍDOS

João Leitão, Sérgio Duarte, Pedro Camponês

(baseado nos slides de Nuno Preguiça)

Aula 12: Capítulo 6

(Introdução ao Teste e) Avaliação de Sistemas
Distribuídos

TESTE E AVALIAÇÃO DE SISTEMAS DISTRIBUÍDOS

Dois objetivos:

- Avaliação de propriedades funcionais
 - O sistema implementa a funcionalidade pretendida?
- Avaliação de propriedades não funcionais
 - Desempenho, escalabilidade, tolerância a falhas, etc.

TESTE E AVALIAÇÃO DE SISTEMAS DISTRIBUÍDOS

Dois objetivos:

- **Avaliação de propriedades funcionais**
 - O sistema implementa a funcionalidade pretendida?
- Avaliação de propriedades não funcionais
 - Desempenho, escalabilidade, tolerância a falhas, etc.

TESTES FUNCIONAIS

Objetivo: verificar que o sistema distribuído efetua as funcionalidades definidas na sua especificação.

Qual a diferença face a sistemas não distribuídos?

É necessário ter em atenção a existência de múltiplos componente independentes, e as falhas que podem surgir na comunicação e nos próprios componentes.

TESTES FUNCIONAIS: PASSO 0

Todo o código desenvolvido deve passar pelos mecanismos normais de teste, independentemente de ser parte dum sistema distribuído: e.g. *unit tests*, etc.

- ***Unit test***: método de teste de software em que os módulos dum programa são testados de forma independente para verificar se estão corretos. Um módulo pode ser uma classe, por exemplo.
- Alguns sistemas para implementar *unit tests* em Java: JUnit, Mockito

JUNIT : EXEMPLO

```
public class UsersResourceTest {  
  
    private final User u = new User("jleitao", "João Leitão", "jc.leitao@fct.unl.pt", "pwd");  
  
    @Test  
  
    public void simpleInsert() {  
        try {  
  
            UsersResource resource = new UsersResource();  
  
            String result = resource.createUser(u);  
            assertEquals(u.getUserId(), result);  
  
            User u0 = resource.getUser(u.getUserId(), u.getPassword());  
  
            assertEquals(u.getUserId(), u0.getUserId());  
            assertEquals(u.getFullName(), u0.getFullName());  
            assertEquals(u.getEmail(), u0.getEmail());  
            assertEquals(u.getPassword(), u0.getPassword());  
            assertEquals(u.getAvatarUrl(), u0.getAvatarUrl());  
            assertEquals(u.toString(), u0.toString());  
  
        } catch (Exception e) {  
            fail(e.getMessage());  
        }  
    }  
}
```

TESTES FUNCIONAIS: *TESTES UNITÁRIOS*

Testes unitários de componentes distribuídos: testar a funcionalidade dos componentes distribuídos quando funcionam isoladamente (se possível).

Como fazer: definir um conjunto de cenários que testem as diferentes possibilidades dos inputs, não só para utilizações que terminam em sucesso mas também para utilizações que terminam em erros.

EXEMPLO: TESTES UNITÁRIOS NO TRABALHO PRÁTICO

Objetivo: testar cada componente isoladamente, considerando situações com e sem concorrência.

Não completamente possível devido à forte ligação entre os serviços Users e Content por exemplo.

Cenários definidos – e.g.:

- Cria novo utilizador; verifica que utilizador existe.
- Cria novo utilizador; remove utilizador; verifica que utilizador não existe.
- Cria novo utilizador; altera dados do utilizador; verifica que dados foram atualizados.
- Verifica que devolve erro apropriado ao aceder a utilizador não existente.
- ...

EXEMPLO: TESTES UNITÁRIOS COM CONCORRÊNCIA

Objetivo: testar cada componente isoladamente, considerando situações com concorrência.

Cenários definidos – e.g.:

- Cria múltiplos utilizadores concorrentemente; verifica que todos os utilizadores existem.
- Cria e remove utilizadores concorrentemente; verifica que todos os utilizadores que devem existir existem.
- ...

TESTES FUNCIONAIS: *TESTES DE INTEGRAÇÃO*

Testes de integração de componentes distribuídos: testar a funcionalidade dos componentes distribuídos quando se integram vários componentes, ainda sem considerar a parte da distribuição e as falhas (se possível).

Como fazer: definir um conjunto de cenários que testem as diferentes possibilidades dos inputs que levem à interação dos componentes, não só para utilizações que terminam em sucesso mas também para utilizações que terminam em erros.

EXEMPLO: TESTES DE INTEGRAÇÃO NO TRABALHO PRÁTICO

Objetivo: testar a integração de componentes no sistema, ainda sem considerar a parte da distribuição e as falhas.

No trabalho, os componentes são distribuídos, pelo que se testava a integração de componentes distribuídos sem falhas.

Cenários definidos – e.g.:

- Criar um Post; verificar que o Post foi criado ou não consoante os parâmetros.
- Criar um Post e removê-lo; verifica que o Post não existe.
- Remover um utilizador; verifica que os upVotes/downVotes desse utilizador foram removidos tentando aceder à contagem dos mesmos nos Post afetados; verificar que os Posts desse utilizador deixaram de ter um utilizador definido como criador...

EXEMPLO: TESTES DE INTEGRAÇÃO COM CONCORRÊNCIA NO TRABALHO PRÁTICO

Objetivo: testar a integração de componentes no sistema, considerando situações de concorrência mas ainda sem considerar a parte da distribuição e as falhas.

No trabalho, os componentes são distribuídos, pelo que se testava a integração de componentes distribuídos sem falhas.

Cenários definidos – e.g.:

- Concorrentemente criar um Post e remover utilizadores.

TESTES FUNCIONAIS: *TESTES FUNCIONAIS*

Testes funcionais de sistemas distribuídos: testar a funcionalidade dos diversos componentes/serviços quando interagem entre si.

Neste momento é necessário considerar o modelo de falhas, testando o comportamento quando não ocorrem falhas e na presença de falhas.

Como fazer: definir um conjunto de cenários que testem as diferentes possibilidades dos inputs que levem à interação dos componentes, não só para utilizações que terminam em sucesso mas também para utilizações que terminam em erros.

Definir um conjunto de cenários que considerem falhas.

EXEMPLO: TESTES FUNCIONAIS NO TRABALHO PRÁTICO

Objetivo: testar o funcionamento do sistemas como um todo, considerando também cenários de falhas.

Cenários definidos – e.g.:

- Criação e acesso a um Post quando existe uma falha de comunicação de curta duração entre os servidores.
- Criação e acesso a um Post quando existe uma falha de comunicação de longa duração entre os servidores.
- ...

TESTES: ALGUNS DESAFIOS

O resultado de algumas operações não é determinista – e.g. timestamp de criação do Post.

- Estes valores não devem ser verificados e deve-se usar o valor retornado pelo servidor para o estado interno do sistema de teste.

Alguns resultados são não-deterministas devido à ordem da execução de operações concorrentes, mas é importante verificar os valores – e.g. o resultado duma pesquisa.

- Verificar os valores independentemente da ordem.

TESTES: ALGUNS DESAFIOS (2)

A execução concorrente de múltiplas operações pode levar a múltiplos estados possíveis.

Exemplos: Não é possível prever o valor final quando se executam concorrentemente múltiplas operações de atualização do “fullName” dum utilizador.

O teste dos resultados deve aceitar como válido qualquer um dos resultados/estados possíveis. Nem sempre é fácil de prever.

TOOLS PARA TESTES DE SISTEMAS DISTRIBUÍDOS

TLA+ [<https://lamport.azurewebsites.net/tla/tla.html>]

- Linguagem para modelar sistemas (especialmente concorrentes e distribuídos)
- Ferramentas para verificar a correção do modelo

Jepsen [<https://jepsen.io/>]

- Biblioteca para criar programas de teste de sistemas distribuídos.

Functional testing

- Existem algumas ferramentas de *load testing* que também podem ser usadas para fazer testes funcionais (mais no fim do capítulo).

TESTER

Programa para testar trabalho prático.

- 11K+ LOC

Arquitetura base:

- Inicia os servidores em “containers” independentes usando o sistema Docker;
- Mantém internamente uma “cópia” do estado do sistema, i.e., para cada servidor/serviço mantém o estado do serviço e disponibiliza as operações.

TESTER: OPERAÇÕES

Para cada operação do sistema, define-se método que executa os seguintes passos:

1. Executa operação no serviço “real”;
 - Usa resultado da execução para lidar com o não determinismo – e.g. identificador do Post e timestamp de criação retornado pelo serviço real é adicionado à representação interna do Post: serviço simulado não gera novo identificador.
2. Executa operação no serviço “simulado”;
3. Verifica que resultado é o mesmo (sucesso ou falha).

TESTER: OPERAÇÕES (CONT.)

Como lidar com invocações REST / SOAP / GRPC ?

1. Estado simulado é igual;
2. Diferente objeto para invocação em REST, SOAP e GRPC, mas com a mesma interface – execução da operação permanece idêntica, apenas mudando a inicialização;
3. Tratamento de falhas uniformizado – e.g. exceções SOAP / GRPC transformadas em códigos HTTP REST.

TESTER: OPERAÇÕES (CONT.)

Como lidar com replicação (no serviço de Content)? *(não aparece no primeiro trabalho prático)*

1. Estado simulado é igual e não é necessário manter as várias réplicas – apenas uma;
2. Diferentes objetos para invocação para cada uma das réplicas;
3. Pode-se dirigir pedido a uma réplica específica ou pedir para executar em todas.

TESTER: TESTES COM FALHAS DE REDE

Falhas de rede simuladas usando iptables – programa que permite configurar as regras de filtragem de pacotes no Linux (sistema usado nos *containers*).

Desafios

- Como garantir que falhas levam a exceções nos programas? Tempo de falhas configurado com base nos timeouts definidos nos programas a testar, de forma a ser superior ao tempo de timeout.
- Como se tenta inferir se programa está a tratar dos pedidos assincronamente? Se a resposta ao pedido é anterior ao fim da falha (*não aparece no primeiro trabalho prático*).

TESTER: TESTES COM FALHAS DOS SERVIDORES

As falhas dos servidores são simuladas terminando e reiniciando os containers em que os servidores estão a executar *(não aparece no primeiro trabalho prático)*.

TESTER: TESTES

Cada teste consiste numa sequência de operações.

As operações podem executar sequencialmente ou concorrentemente.

TESTE E AVALIAÇÃO DE SISTEMAS DSITRIBUÍDOS

Dois objetivos:

- Testes funcionais
- **Avaliação de propriedades não funcionais**

QUE PROPRIEDADE NÃO FUNCIONAIS

Num sistema distribuído, as propriedades não funcionais tipicamente interessantes são:

- **Latência:** tempo para executar um pedido
- **Performance (*throughput*):** número de pedidos que o sistema consegue executar (por unidade de tempo)
- **Escalabilidade:** como o número de pedido que o sistemas consegue tratar aumenta quando o número de servidores aumenta

MUITO IMPORTANTE

A **avaliação** que se faz a um sistema **deve servir para responder a uma pergunta** – não apenas para obter um número ou um gráfico para decorar um relatório 😊 !!!

De seguida apresentam-se exemplos de questões que é comum colocar.

PERGUNTA 1:

Num sistema com um servidor, **qual a latência observada pelos clientes em função da carga do servidor?**

Como testar?

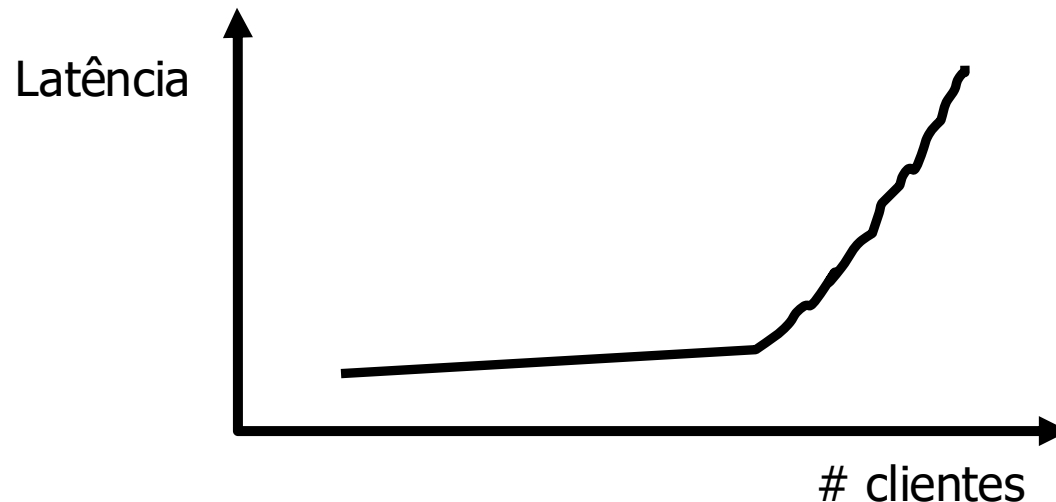
Fazer uma sequência de experiências, aumentando o número de clientes entre cada experiência até ao ponto que a latência começa a aumentar muito.

Repetir cada experiência K vezes (e.g. $K = 3$).

PERGUNTA 1: REPORTAR RESULTADOS

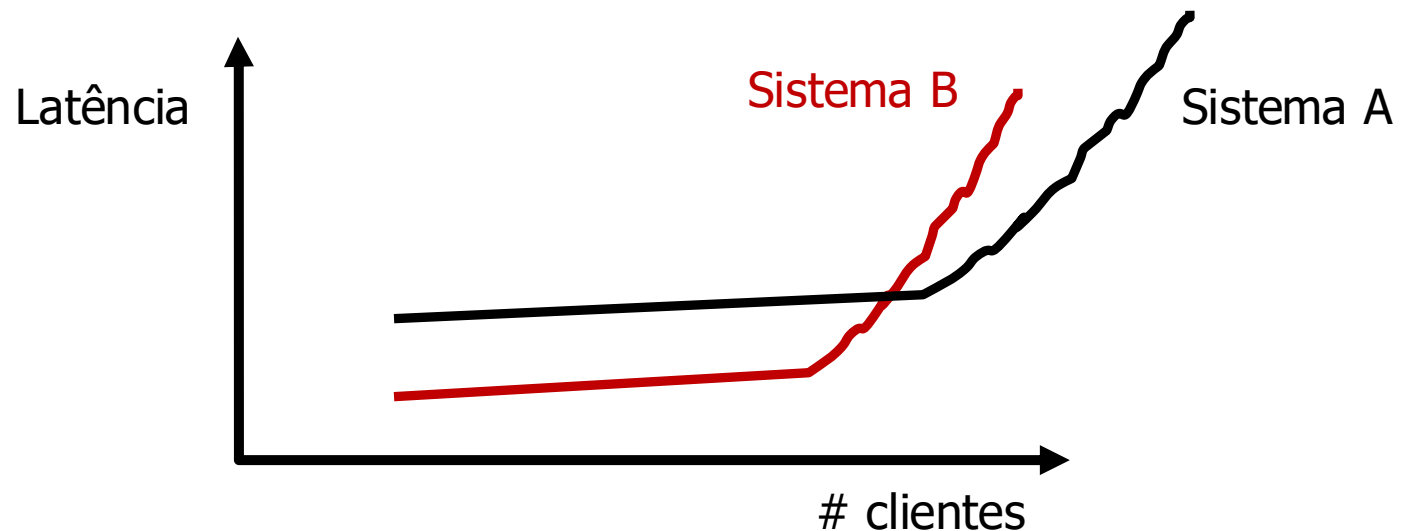
Quais são os resultados esperados quando o servidor executa operações concorrentemente?

- latência permanece constante enquanto o servidor não está em sobrecarga;
- latência aumenta quando o servidor se aproxima da sobrecarga.



PERGUNTA 1: COMPARATIVO

Muitas vezes, o importante na resposta não são os valores absolutos mas a comparação com os valores de outro sistema. Nesse caso, devemos executar as experiências com os dois sistemas e apresentar os resultados. [isto aplica-se também a todas as perguntas seguintes]



PERGUNTA 2:

Num sistema com um servidor (ou N servidores), **qual o número máximo de pedidos que o servidor pode suportar?**

Como testar?

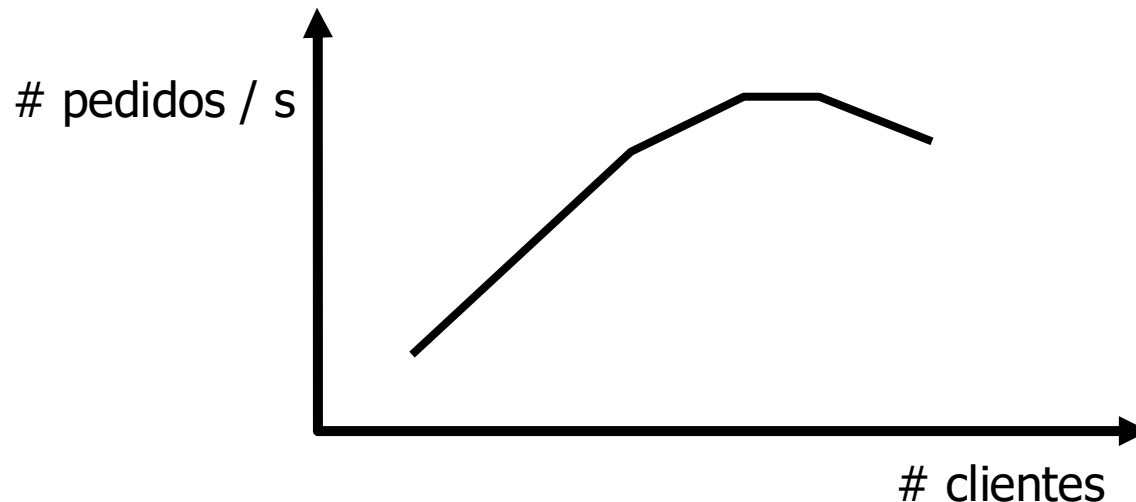
Fazer uma sequência de experiências, aumentando o número de clientes entre cada experiência, até ao ponto em que o número de pedidos deixa de aumentar.

Repetir cada experiência K vezes (e.g. $K = 3$).

PERGUNTA 2: REPORTAR RESULTADOS

Quais são os resultados esperados quando o servidor executa operações concorrentemente?

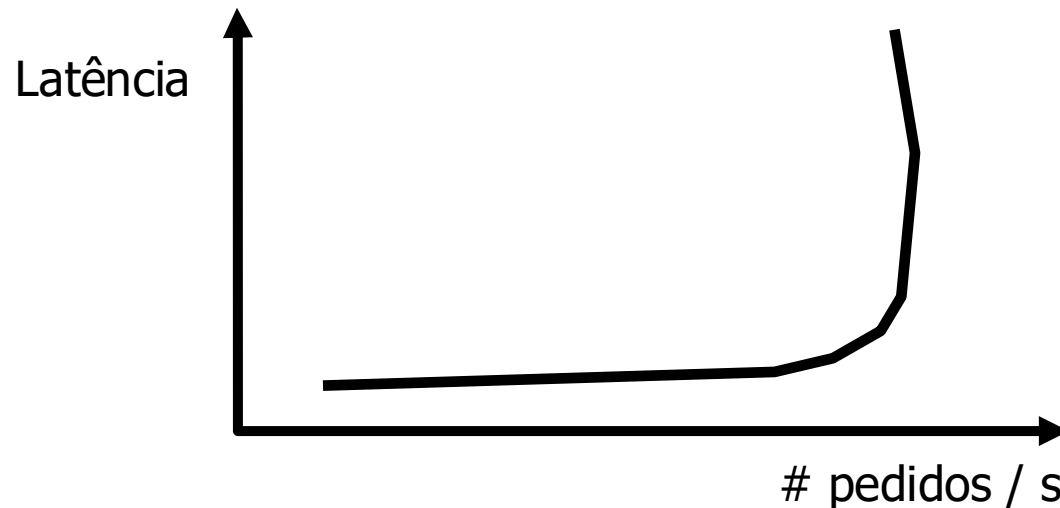
- Número de pedidos aumenta inicialmente linearmente e depois mais lentamente até ao ponto em que deixa de aumentar; após esse ponto é comum o número de pedidos diminuir.



PERGUNTA 2: REPORTAR RESULTADOS

Podemos reportar a latência e o throughput num só gráfico?

Sim: cada ponto corresponderá aos resultados com um dado número de clientes e fazemos uma linha ligando os vários pontos.

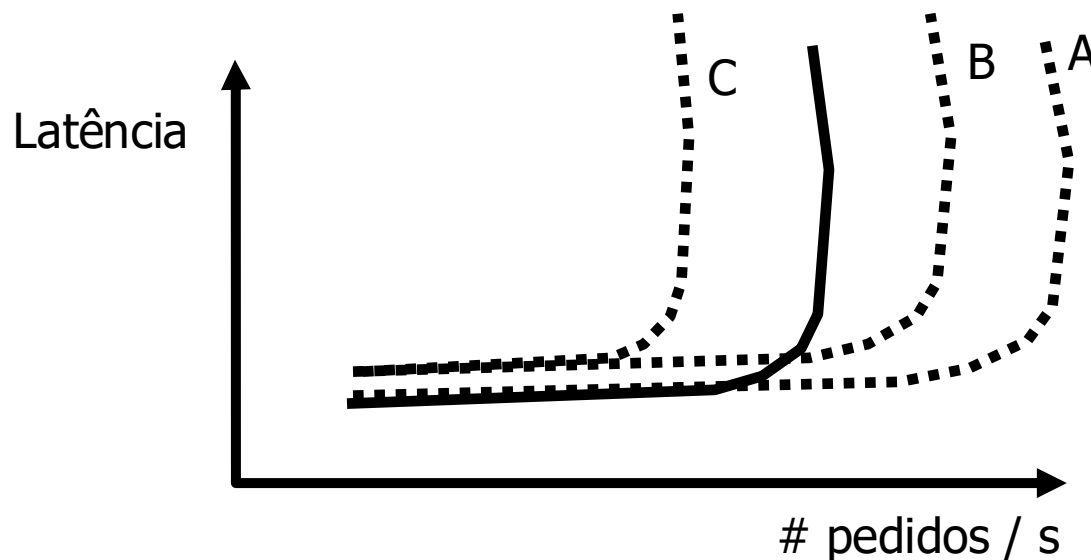


PERGUNTA 2: REPORTAR RESULTADOS

Podemos reportar a latência e o throughput num só gráfico?

Sim: cada ponto corresponderá aos resultados com um dado número de clientes e fazemos uma linha ligando os vários pontos.

Claro que os resultados de um sistema podem ser pouco indicativos apenas por si...



PERGUNTA 3:

Como varia a performance/latência quando se aumenta o número de servidores?

Como testar?

Fazer uma sequência de experiências, aumentando o número de clientes entre cada experiência, até ao ponto em que o número de pedidos deixa de aumentar.

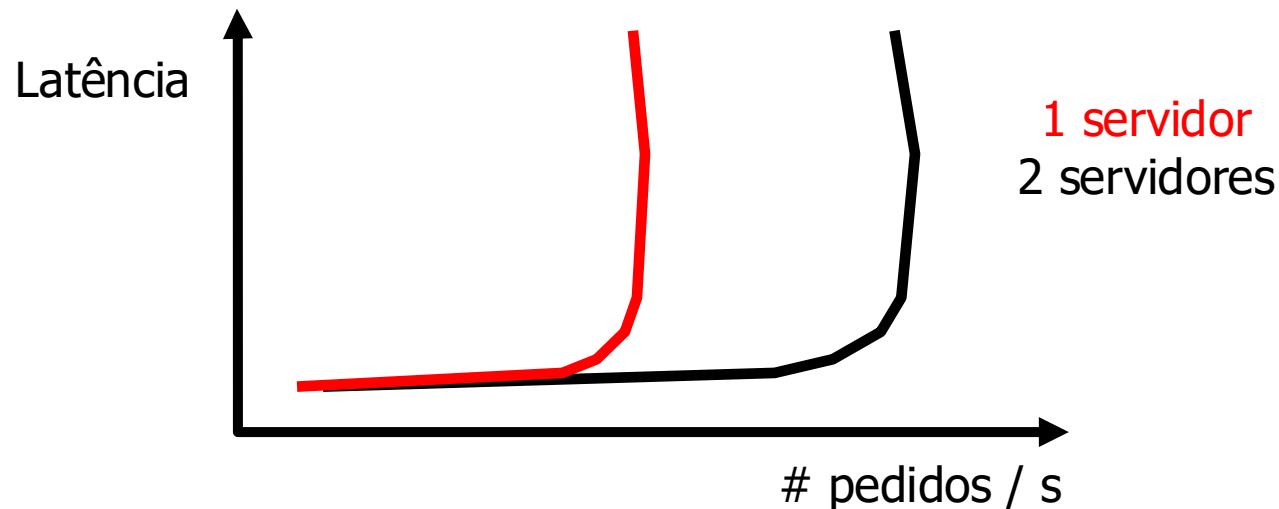
Repetir cada experiência K vezes (e.g. $K = 3$).

Repetir o passo anterior aumentando o número de servidores entre cada passo.

PERGUNTA 3: REPORTAR RESULTADOS

Quais são os resultados esperados quando os servidores executam operações concorrentemente?

- Latência permanece constante enquanto os servidores não estão em sobrecarga;
- Ponto de sobrecarga é maior com mais servidores.



PERGUNTA 4:

No contexto X (um dos anteriores), qual a latência/throughput das diferentes operações do sistema?

Como varia a latência/throughput do sistema com diferentes workloads?

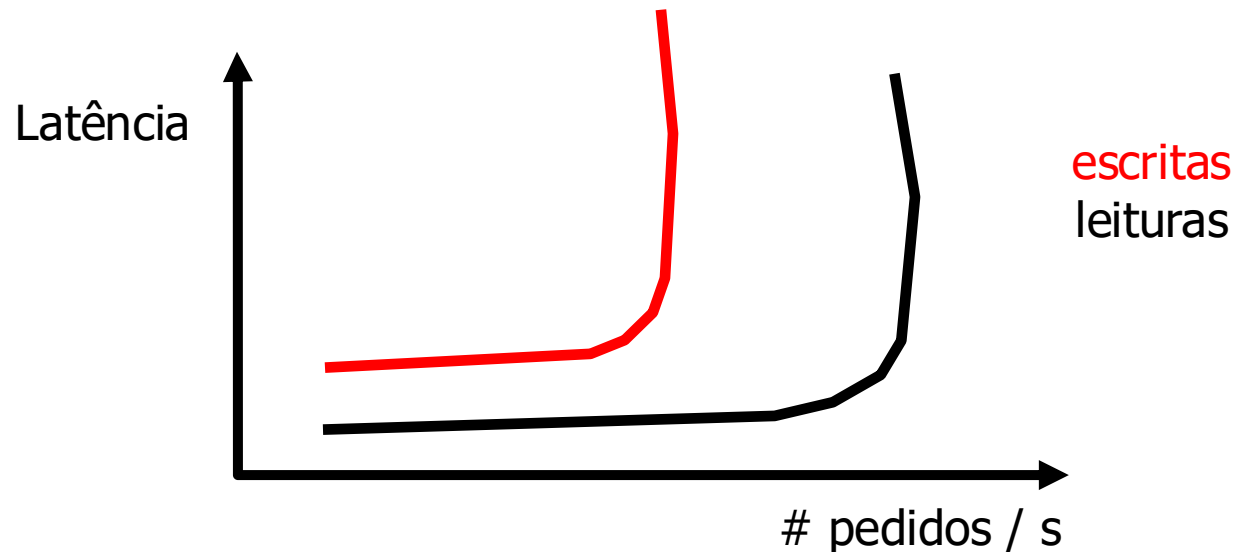
Como testar?

Como nas perguntas anteriores, mas comparando diferentes operações/workloads.

PERGUNTA 4: REPORTAR RESULTADOS

Quais são os resultados esperados?

- A performance de uma operação tende a ser pior quanto mais mensagens e leituras/escritas originais; as escritas são normalmente mais lentas que as leituras.



PERGUNTA 5:

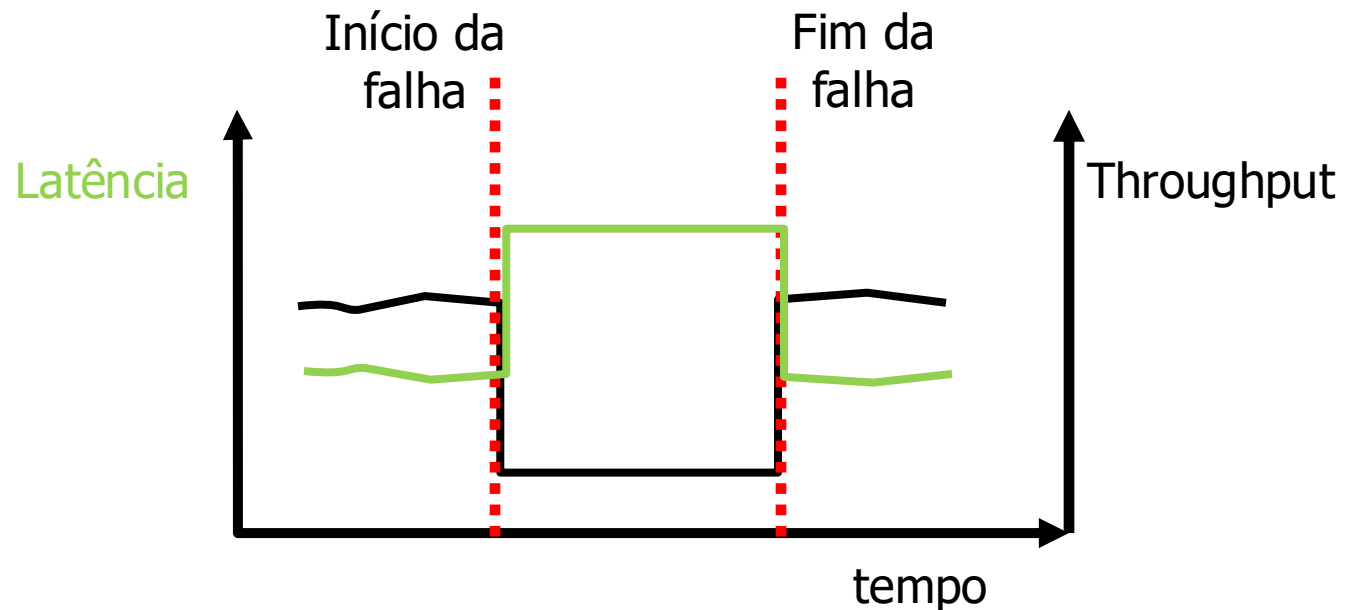
Qual o comportamento do sistema, latência/throughput, na presença de falhas?

Como testar: para um dado setup, simular falhas durante um período limitado da experiência. Obter resultados em janelas de tempo antes, durante e depois das falhas.

PERGUNTA 5: REPORTAR RESULTADOS

Quais são os resultados esperados?

- A latência do sistema aumenta durante o período de falhas e o throughput diminui. Após a falha voltam ao normal, podendo haver alguma oscilação.



TOOLS PARA AVALIAÇÃO DE SISTEMAS DISTRIBUÍDOS

“Load testing” é o teste de sistemas colocando carga nos sistemas e medindo o seu comportamento. Existem muitas ferramentas de “load testing” de sistemas distribuídos.

Exemplos

JMeter [<https://jmeter.apache.org/>]

- Aplicação para “Load testing”. Inicialmente de serviços web, mas atualmente de outros serviços.
- Relativamente complexo de definir testes.

Artillery [<https://artillery.io/>]

- Ferramenta de load testing e functional testing de web services REST.

TOOLS PARA AVALIAÇÃO DE SISTEMAS DISTRIBUÍDOS (2)

As ferramentas de “load testing” permitem muitas vezes fazer “functional testing”, mas o suporte é tipicamente bastante primitivo – quem define o teste tem de definir qual o resultado esperado para cada operação, o que tipicamente não é simples em testes que sejam feitas muitas operações ou operações aleatórias.