COMPUTAÇÃO NA NUVEM
ISEL – LEIRT / LEIC

Características e Desafios
da Computação Distribuída

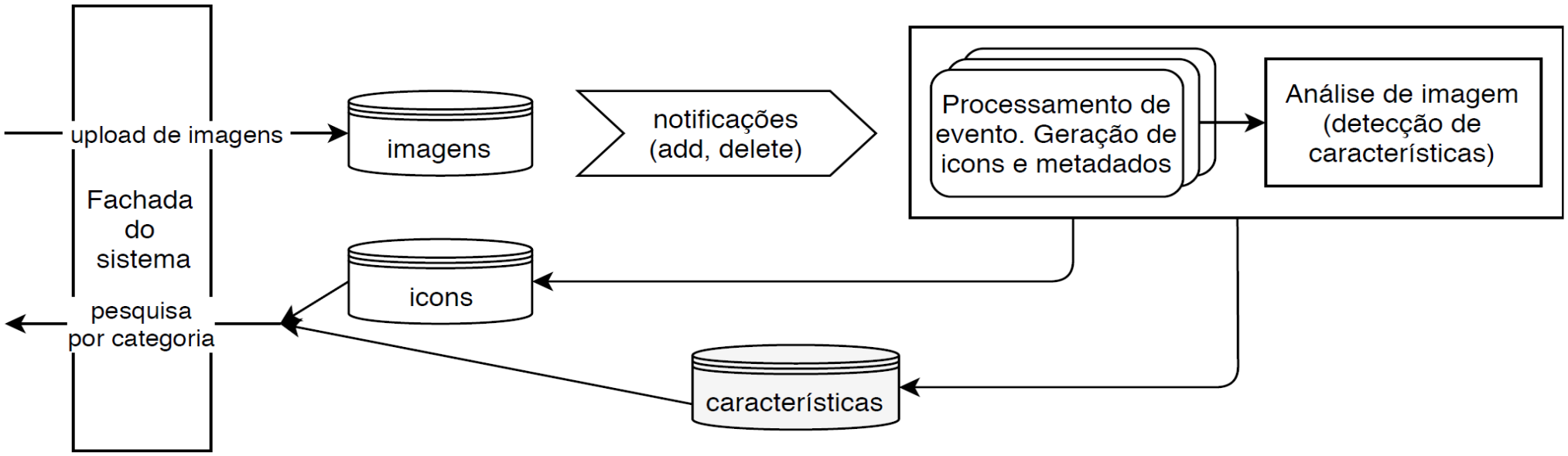
José Simão jsimao@cc.isel.ipl.pt ; jose.simao@isel.pt

Luís Assunção lass@isel.ipl.pt ; luis.assuncao@isel.pt

Sumário

- Características e desafios principais
- Teorema CAP (*Consistency, Availability, Partitions*)
- Disponibilidade (*Availability*) num sistema distribuído
- As 8 Falácias da Computação Distribuída
- Latência & Cloud

Um exemplo



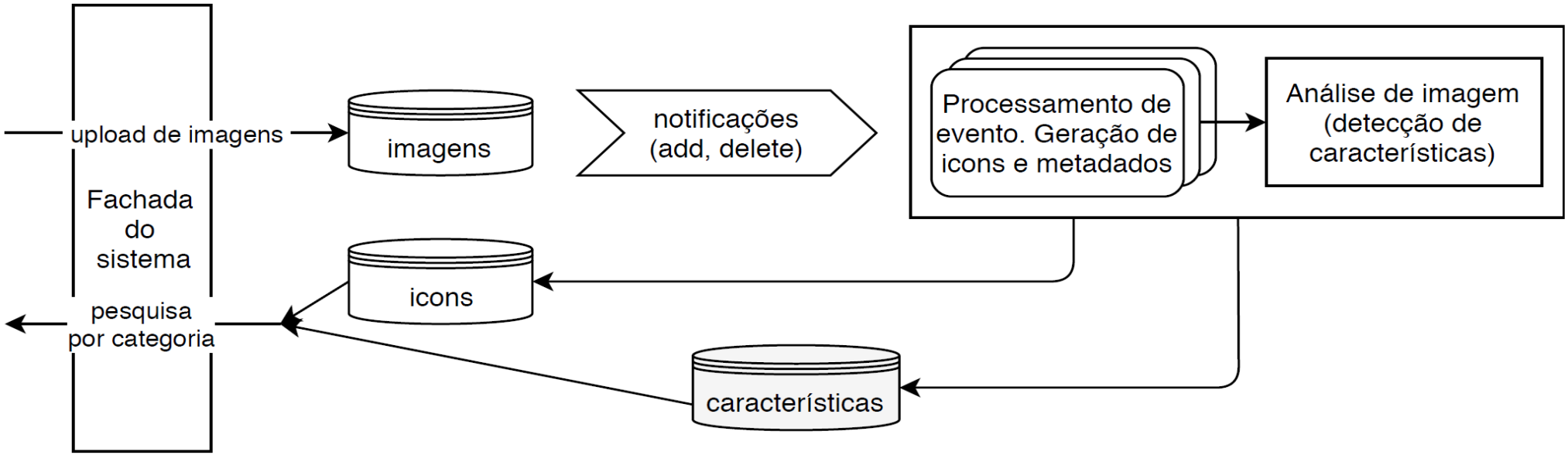
- Sistema para *upload* e pesquisa de imagens
  - *Upload* de imagem completa
  - Identificação de etiquetas
  - Geração e representação em miniatura da imagem
  - Pesquisa por características

Um exemplo

Segurança -
Autenticação
Autorização

Processamento **concorrente** de eventos

Escalabilidade do sistema



Disponibilidade e **qualidade** de serviço

Sistemas **abertos** ou com interface proprietária

Processamento de **falhas** na análise

Características e Desafios Principais

- Heterogeneidade
- Sistemas Abertos
- Segurança
- Escalabilidade (*Scalability*)
- Tratamento de Falhas
- Concorrência
- Transparência
- Qualidade de Serviço (*QoS - Quality of Service*)

Heterogeneidade

- Diferentes topologias de rede;
- Hardware de computadores;
- Sistemas Operativos;
- Linguagens de Programação;
- Bases de dados com diferentes paradigmas
- Implementações em diferentes ambientes, usando diferentes *middlewares*

Através de várias organizações de normalização é, hoje, possível ter disponível um conjunto de normas e iniciativas abertas, que facilitam a interoperabilidade entre sistemas heterogéneos:

- TCP/IP, HTTP, HTML, SOAP, XML, JSON, SQL, ODBC, etc.

Sistemas Abertos (*Openess*)

- Dada a heterogeneidade de Hardware e Software, a especificação e documentação das interfaces deve ser do domínio público, incluindo implementações de referência (Sockets, plataformas Java);
- Mecanismos de comunicação e publicação dos serviços para acesso aos recursos partilháveis baseados em normas aceites pela comunidade e pela indústria (ex. DNS, REST, SOAP e WSDL);
- A interoperabilidade (*openess*) assenta no esforço de grupos Nacionais e Internacionais de normalização e de cooperação, como são exemplo os RFCs (*Request For Comments*) de muitos protocolos (ex. TCP/IP, HTTP) e *World Wide Web Consortium* (W3C)
- Dada a inexistência de interoperabilidade entre fornecedores, existem neste momento várias iniciativas relacionadas com Cloud:
  - Comunidade científica: Open Commons Consortium [OCC]; Open Science Data Cloud (OSDC);
  - Infraestrutura: *Java Multi-Cloud Toolkit*; *Apache CloudStack*.

[OCC] cloud computing and data commons infrastructure to support scientific, medical, health care and environmental research

Segurança

- Privacidade - (confidencialidade) - protecção contra acessos não autorizados;
- Integridade - Protecção contra alterações e corrupção dos dados;
- Autenticação – Garantia de que os interlocutores são quem dizem ser;
- Autorização – Os utilizadores autenticados podem ter permissões diferentes para as diferentes actividades;
- Disponibilidade – Protecção contra interferências no acesso ao serviço;

➢ Os desafios:

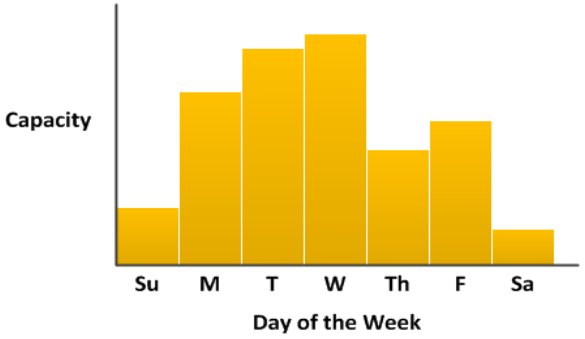
  - Protecção contra ataques de negação do serviço (*denial of service*), por exemplo, "bombardeamento" de um serviço com pedidos, impossibilitando outros utilizadores de o usar;
  - Segurança de código móvel;
  - Roubo de identidades;

Escalabilidade (*Scalability*)

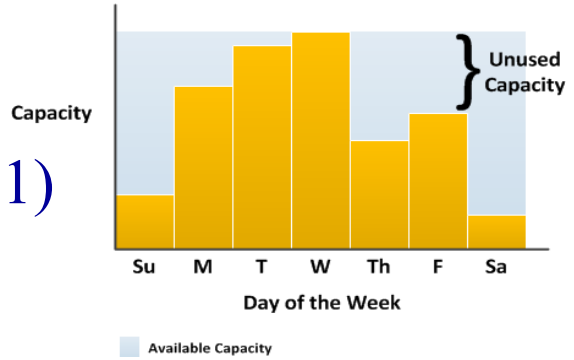
- Um sistema tem a propriedade de escalabilidade se permanece efectivo (disponível, com desempenho e tempos de resposta aceitáveis) quando é aumentado um número aceitável de recursos e utilizadores;
- Os sistemas distribuídos com escalabilidade têm os seguintes desafios:
  - Custo controlado de recursos físicos (computadores, memória principal, memória secundária, periféricos, …)
  - Avaliação antecipada da evolução do sistema (volumes de informação, engarrafamentos, espaço de nomes, saturação de endereços IPv4/IPv6)
  - Controlo da perda de desempenho – Quais os indicadores e métricas associadas (Nº transações por segundo, Volume de dados, Tempos de resposta, Percentagem de CPU, etc.)

Escalabilidade vs Elasticidade

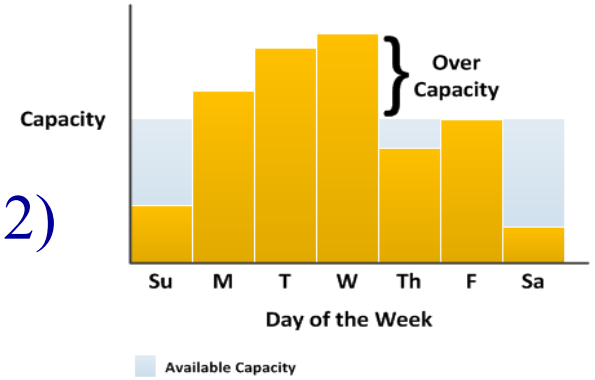
capacidade
versus
necessidades



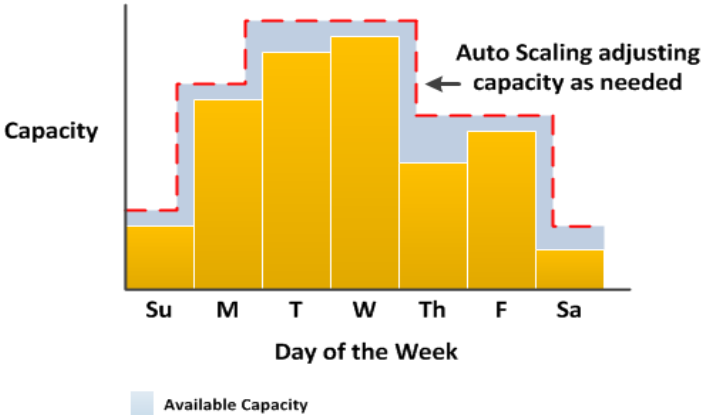
1)



2)



Elasticidade: Adapta a capacidade em função das necessidades



https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-benefits.html

- Expande ou reduz recursos dinamicamente, baseado em condições:
  - Aumento de utilização de CPU de VMs;
  - Num determinado dia e hora;
  - Número de utilizadores concorrentes

Tratamento de falhas (*Fault tolerance*)

- As falhas num sistema distribuído são parciais, isto é, alguns componentes falham, enquanto outros continuam em função. Detetar e tratar falhas parciais é difícil;
- Técnicas para lidar com falhas:
  - Redundância do hardware (clustering de servidores, RAID (*Redundant Array of Independent Disks*), etc.)
  - Recuperação de erros por parte do software: retransmissão de mensagens e replicação de dados em diferentes localizações;
  - Um utilizador deverá poder continuar a sua atividade numa estação de trabalho alternativa;
  - Replicação de recursos e servidores;
  - Reposição de estados anteriores após uma falha (*rollback*);
  - A disponibilidade (*Availability*) é normalmente maior num sistema distribuído, com a facilidade em disponibilizar recursos alternativos, potenciado pelas tecnologias de virtualização

Falhas bizantinas

- Falhas arbitrárias usadas para descrever os piores cenários de falhas, onde diferentes observadores podem observar diferentes comportamentos num sistema

*"**Byzantine generals problem**"*

Consideremos unicamente 3 generais A, B e C em que A é traidor;

- ✓ A diz a B ⟶ *Atacar*
- ✓ A diz a C ⟶ *Retirar*
- ✓ B diz a C ⟶ O general A mandou *Atacar*

C não pode concluir quem é traidor. Se A ou B?

Lamport, Shostack, and Peace, provaram [1] que o problema só tem solução se,

$$N \geq 3t + 1$$

N – número de generais, t – número de generais traidores.

isto é, na presença de falhas é necessário recorrer a algoritmos distribuídos de geração de consensos por maioria.

[1] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

Falhas Bizantinas

- Um sistema é tolerante a falhas bizantinas, se continua a funcionar mesmo quando algumas partes do sistema não funcionam ou não obedecem a protocolos estabelecidos;
  - Ambientes aeroespaciais, sistemas de controlo de tráfego aéreo;
  - Sistemas colaborativos (multi-organização), onde não exista uma autoridade central: ex: *bitcoin*, *blockchain*;
- Na prática, devido à complexidade, não é comum suportar falhas bizantinas, recorrendo a técnicas de ter uma autoridade que uma vez eleita decide se algum comportamento pode ou não prosseguir;
- Os algoritmos de suporte a falhas bizantinas requerem normalmente consensos de maioria ou mesmo de 2/3 dos participantes:
  - Usando uma aproximação do tipo "generais bizantinos", esta aproximação para detetar *bugs* de software requeria 4 implementações diferentes e esperar que apenas uma tenha bugs!

Concorrência

- Processos executados em concorrência ou paralelo
  - Um processador (core) versus múltiplos processadores (cores)
  - Múltiplos servidores (VMs) distribuídos
- Servidores concorrentes na resposta a clientes (múltiplas *threads* implícitas ou explicitas para atender os pedidos no servidor)
  - Controlo da Concorrência
  - Mecanismos de sincronização, exclusão mútua, gestão de filas de espera (prioridades), Eleição; Chamadas assíncronas, etc.
- Transações Distribuídas

Transparência (i)

- Ao acesso: O acesso a recursos locais e remotos usa operações idênticas. Esconde diferenças entre representação de dados e de convenções de nomeação de recursos (ex: acesso a ficheiros);
- À localização: Permite que recursos em diferentes locais, sejam acedidos do mesmo modo sem conhecimento da sua localização, por exemplo endereços IPs. A nomeação de recursos (por exemplo, URLs com nomes de domínio) tem um papel importante na transparência à localização;
- À migração: Permite a deslocação de recursos ou processos sem afetar o modo de acesso aos mesmos;
- À concorrência: Permite que múltiplos processos concorrentes, usem recursos partilhados, sem interferência, isto é, múltiplos acessos concorrentes devem deixar o recurso num estado consistente. Por exemplo, um objecto/serviço servidor pode processar vários pedidos de vários clientes em concorrência.

(cont.)

Transparência (ii)

- Às réplicas: Uso de múltiplas instâncias (réplicas) de um recurso para melhorar o desempenho, segurança e a tolerância a falhas, sem conhecimento por parte do utilizador ou programador da existência de réplicas. A transparência às réplicas implica transparência à localização, isto é todas as réplicas devem ter o mesmo nome. Exemplos: Bases de Dados distribuídas; *Mirroring Web Sites*
- Às falhas: Disfarce de falhas, permitindo a conclusão de tarefas perante falhas de hardware ou de componentes de software. Mascarar falhas é um dos aspetos mais complexo, ou mesmo impossível, em sistemas distribuídos;
- Ao desempenho: Permite a reconfiguração de um sistema de modo a acompanhar as variações de carga. Por exemplo, Serviços na Cloud de *Auto-scaling*;
- À escalabilidade: Permite a escalabilidade do sistema sem alteração da estrutura do sistema nem dos algoritmos das aplicações

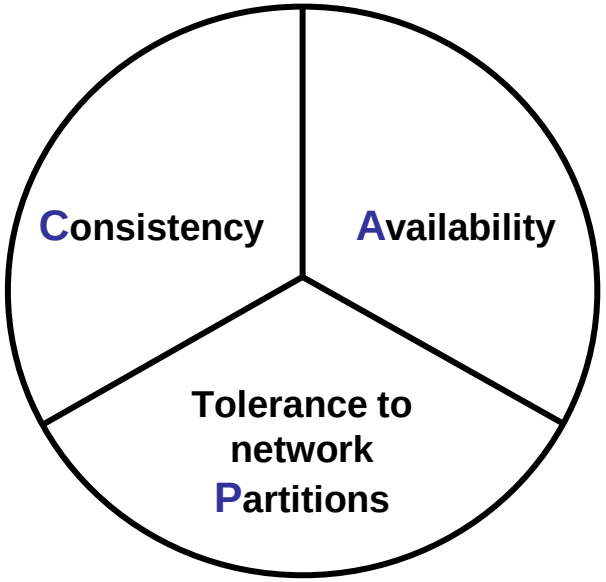
Existem conflitos (*trade-off*) entre altos níveis de transparência e o desempenho

Qualidade de Serviço

- Garantir que os utilizadores têm acesso às funcionalidades de um serviço dentro de limites definidos para determinados indicadores, por exemplo, o tempo de transmissão na distribuição de conteúdos multimédia;
- As principais características, não funcionais, envolvidas que afectam a QoS são a fiabilidade (reliability), segurança, desempenho e a capacidade de reconfigurar o sistema (adaptability).
- A disponibilidade em tempo apropriado, dos recursos computacionais e de comunicação necessários;

- Na Cloud qualidade de serviço (QoS) é importante:
  - Tempos de execução e de resposta;
  - Variações de carga (*workloads*) versus alocação e atribuição dinâmica de recursos, por exemplo VMs;
  - Possibilidade de monitorizar e definir restrições que garantam os níveis de serviços SLA (*Service Level Agreements*) adequados

Teorema CAP?



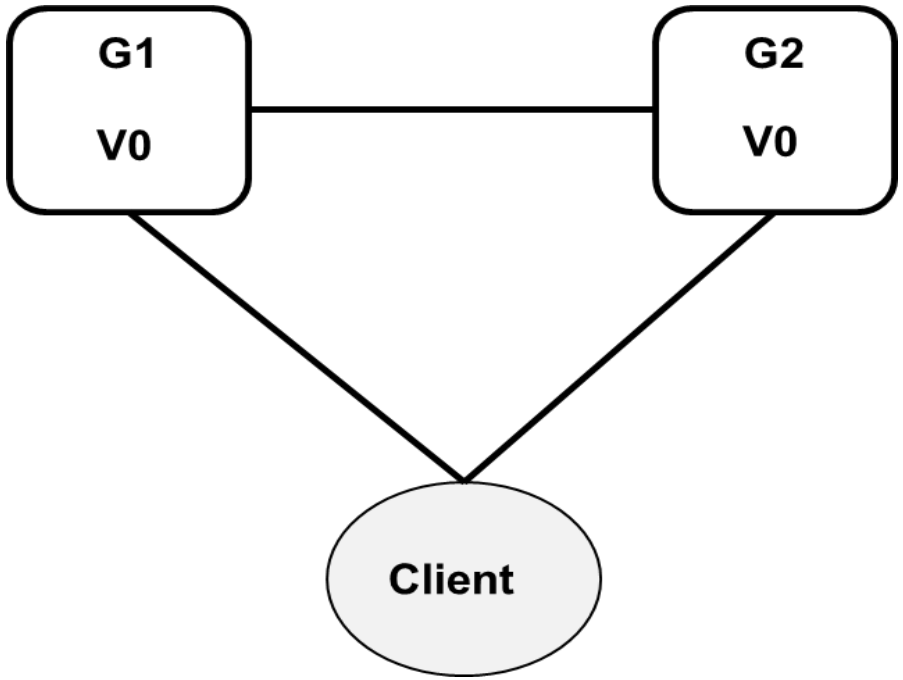
"Num sistema distribuído que partilhe dados só podemos ter 2 das 3 propriedades"

Eric A. Brewer - UC Berkeley, 2000

- *Consistency* – Cada leitura observa a última escrita. O sistema oferece um estado consistente para todos os observadores.
- *Availability* – O sistema continua a funcionar (eventual degradação na qualidade de serviço) na presença de falhas ou de falta de conectividade de alguns nós.
- *Partitions Tolerance* - O sistema continua a funcionar apesar de haver atraso ou falha na entrega de mensagens.

Teorema CAP num sistema distribuído

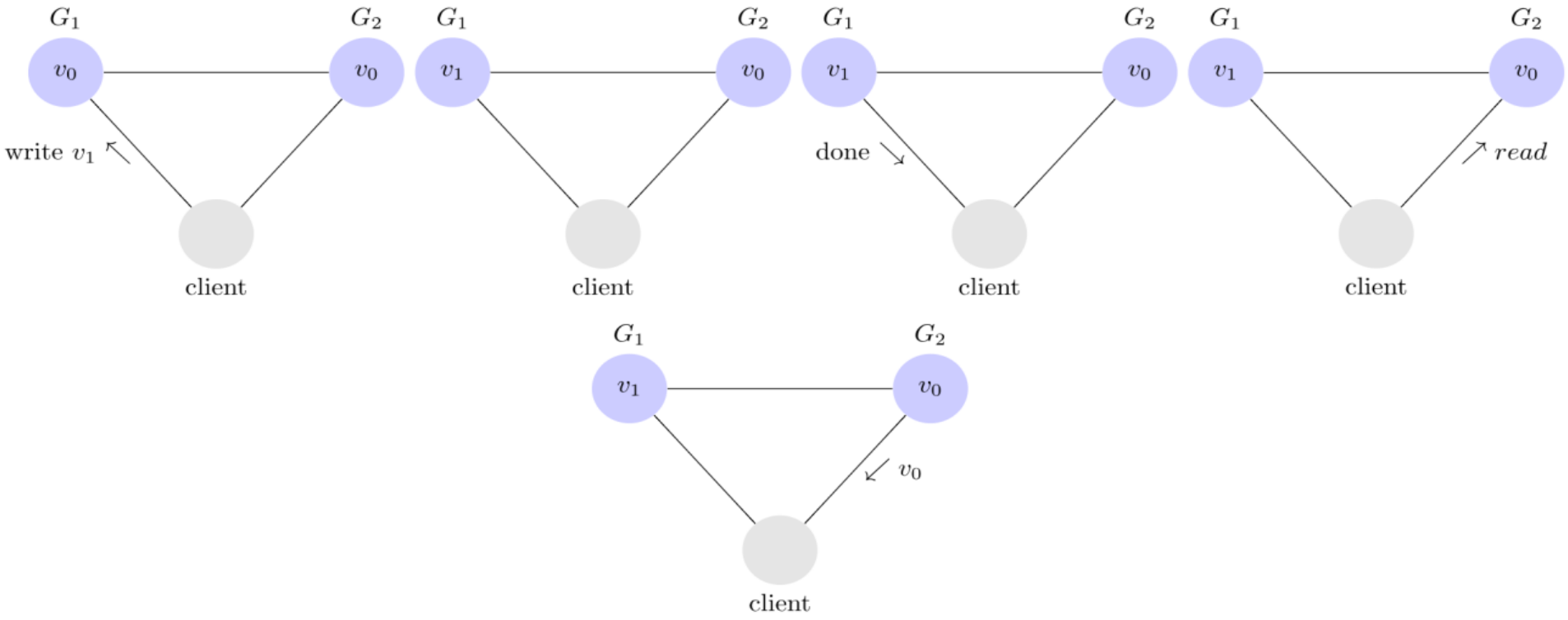
- Um sistema com dois servidores G1 e G2 que comunicam entre si
- O cliente pode comunicar com os dois servidores
- Os servidores têm um mesmo objeto com o valor V0



* Baseado em: *https://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/*

Consistência

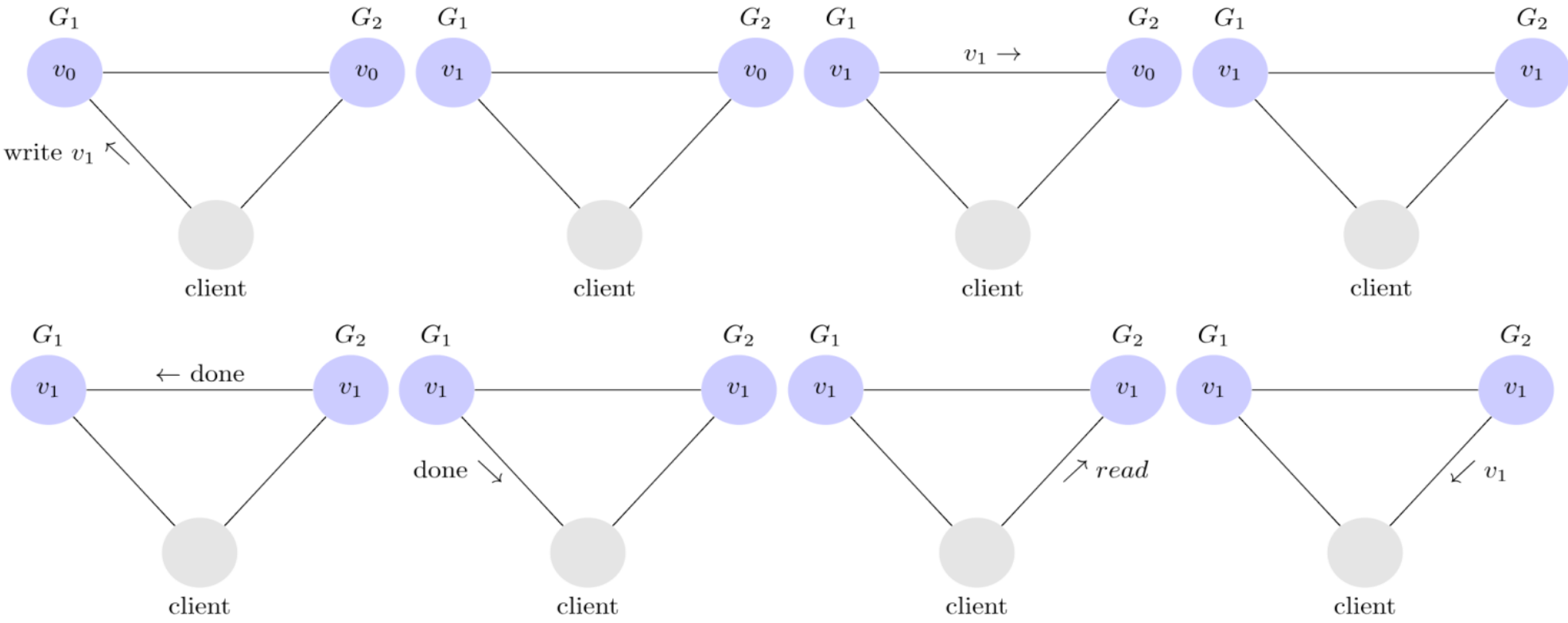
- Uma operação READ devolve sempre o valor da última operação WRITE



Inconsistente

Consistência

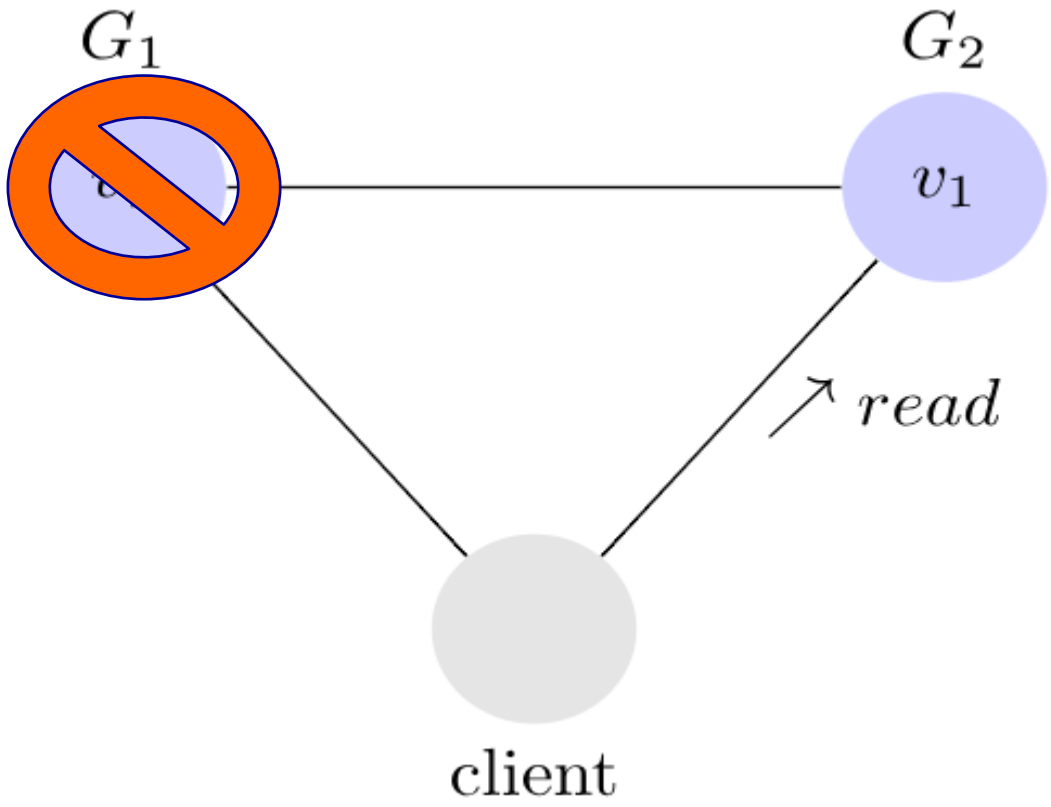
- Uma operação READ devolve sempre o valor da última operação WRITE



Consistente

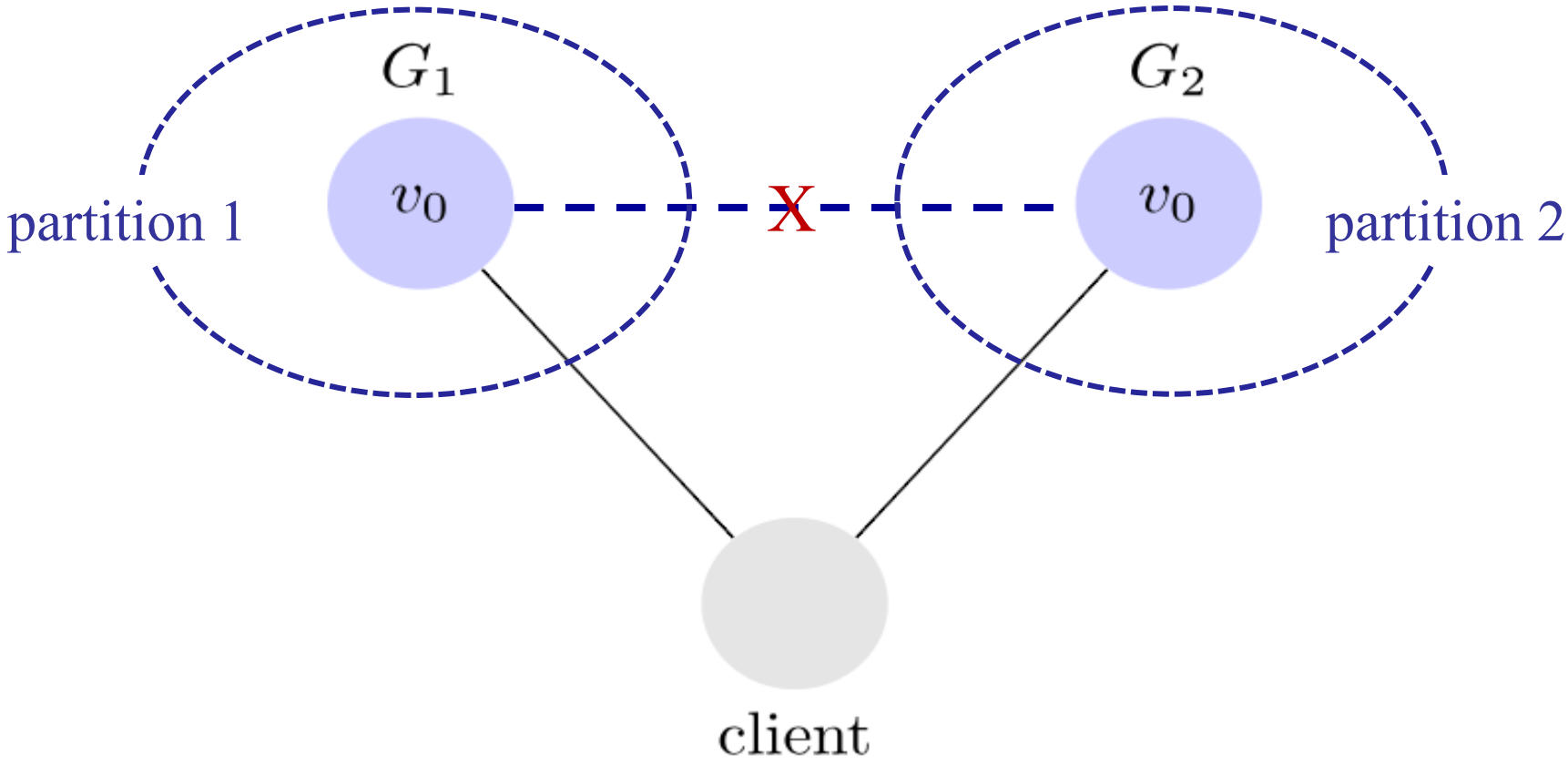
Disponibilidade

- Todos os pedidos têm uma resposta, isto é um dos servidores disponíveis deve responder aos pedidos do cliente



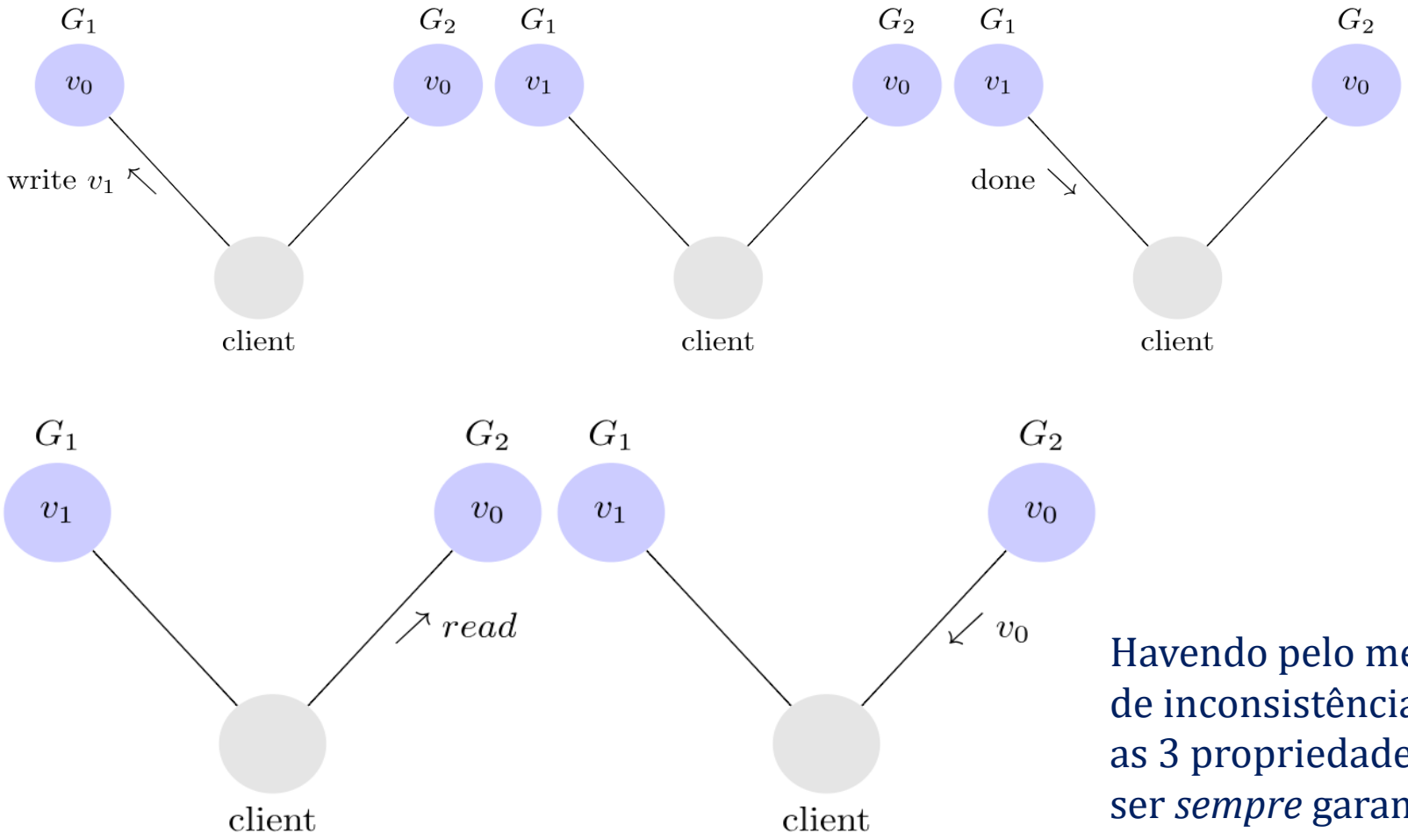
Tolerância à partição

- Tolerância à perda arbitrária de mensagens enviadas entre os nós do sistema (neste caso 2 servidores)



“Prova” do CAP

- Por absurdo, vamos admitir que um sistema tem as 3 propriedades



Havendo pelo menos 1 cenário de inconsistência,
as 3 propriedades não podem ser *sempre* garantidas

Teorema CAP e o desenho de aplicações

- Os arquitetos de aplicações distribuídas, nomeadamente na Cloud, necessitam definir um compromisso, maximizando combinações de consistência (*Consistency*) e disponibilidade (*Availability*), assumindo que existem partições (*Partitions*) e técnicas para lidar com elas;
- Por exemplo, os sistemas de armazenamento de dados NoSQL, baseiam-se em técnicas adequadas para trabalhar com múltiplas partições (centenas ou milhares de computadores ligados em rede), privilegiando a disponibilidade
  - **Semântica BASE (*Basically Available, Soft state, Eventually consistent*).**

Disponibilidade

- *Availability* – razão em percentagem do tempo em que um sistema funciona (*Uptime*) versus o tempo em que o sistema deveria funcionar;

$$\frac{Uptime}{Uptime + Downtime}$$

- *"Five nines"* ou 99.999% é normalmente o objetivo de ambientes críticos, isto é, 5,26 minutos de *downtime* por ano ou cerca de 26 segundos de *downtime* por mês;

- Para ter *"Five nines"* a intervenção humana para recuperação de falhas é irrealista, pelo que as infraestruturas de alta disponibilidade têm de ser capazes de recuperar de falhas automaticamente sem intervenção humana.

As 8 falácias dos sistemas distribuídos

1. A rede é confiável
   - as aplicações precisam de tratar erros e repetir chamadas
2. A latência é zero
   - as aplicações precisam de minimizar pedidos (interações remotas)
3. A largura de banda é infinita
   - as aplicações devem usar o menor *payload* possível para o problema
4. A rede é segura
   - As aplicações têm de garantir segurança na comunicação ponto a ponto
5. A topologia da rede não muda
   - Mudanças afetam latência, largura de banda e destinos
6. Existe um único administrador
   - Interação com diferentes sistemas e políticas de gestão
7. Custo de transporte é zero
   - Na cloud existe muitas vezes um custo monetário para mover dados
8. A rede é homogénea (computadores e rede com mesma configuração)
   - Afeta a fiabilidade, latência e largura de banda

A ler: https://www.simpleorientedarchitecture.com/8-fallacies-of-distributed-systems/

Latência e largura de banda

- Latency – Tempo para transferir dados de um ponto para outro.
- Bandwidth – Quantidade de dados que podemos transferir num determinado tempo.

Latency vs. Bandwidth – Developers vs. Einstein by *Ingo Rammer*

> *"But I think that it's really interesting to see that the end-to-end bandwidth increased by 1468 times within the last 11 years while the latency (the time a single ping takes) has only been improved tenfold. If this wouldn't be enough, there is even a natural cap on latency. The minimum round-trip time between two points of this earth is determined by the maximum speed of information transmission: the speed of light. At roughly 300,000 kilometers per second (3.6 * 10E12 teraangstrom per fortnight), it will always take at least 30 milliseconds to send a ping from Europe to the US and back, even if the processing would be done in real time."*

Exemplo: Latência & Cloud - *File Upload* (1 MB)