Bancos de Dados distribuídos

FRANCISCO PAULO DE FREITAS NETO

F.FREITAS@IFPB.EDU.BR

Introdução

Como discutimos, um dos principais motivos que levaram ao surgimento do NoSQL é a execução em clusters;

Porém, existe uma complexidade associada a isso.

- Só devemos usar se os benefícios forem visíveis

Existem modelos de distribuição que permitem:

- Maior tráfego de leitura/gravação
- Maior disponibilidade

Introdução

Existem principalmente dois modelos de distribuição:

- Replicação: dados replicados em vários nós
- Fragmentação: dados "espalhados" entre vários nós

As técnicas podem ser utilizadas separadamente ou em conjunto.

Introdução

Caso seja possível, **é preferível que se utilize apenas um servidor**;

O banco fica em uma única máquina e toda leitura e gravação fica sob responsabilidade dessa máquina;

Além de mais simples, evita a complexidade que os modelos de distribuição introduzem.

Fragmentação

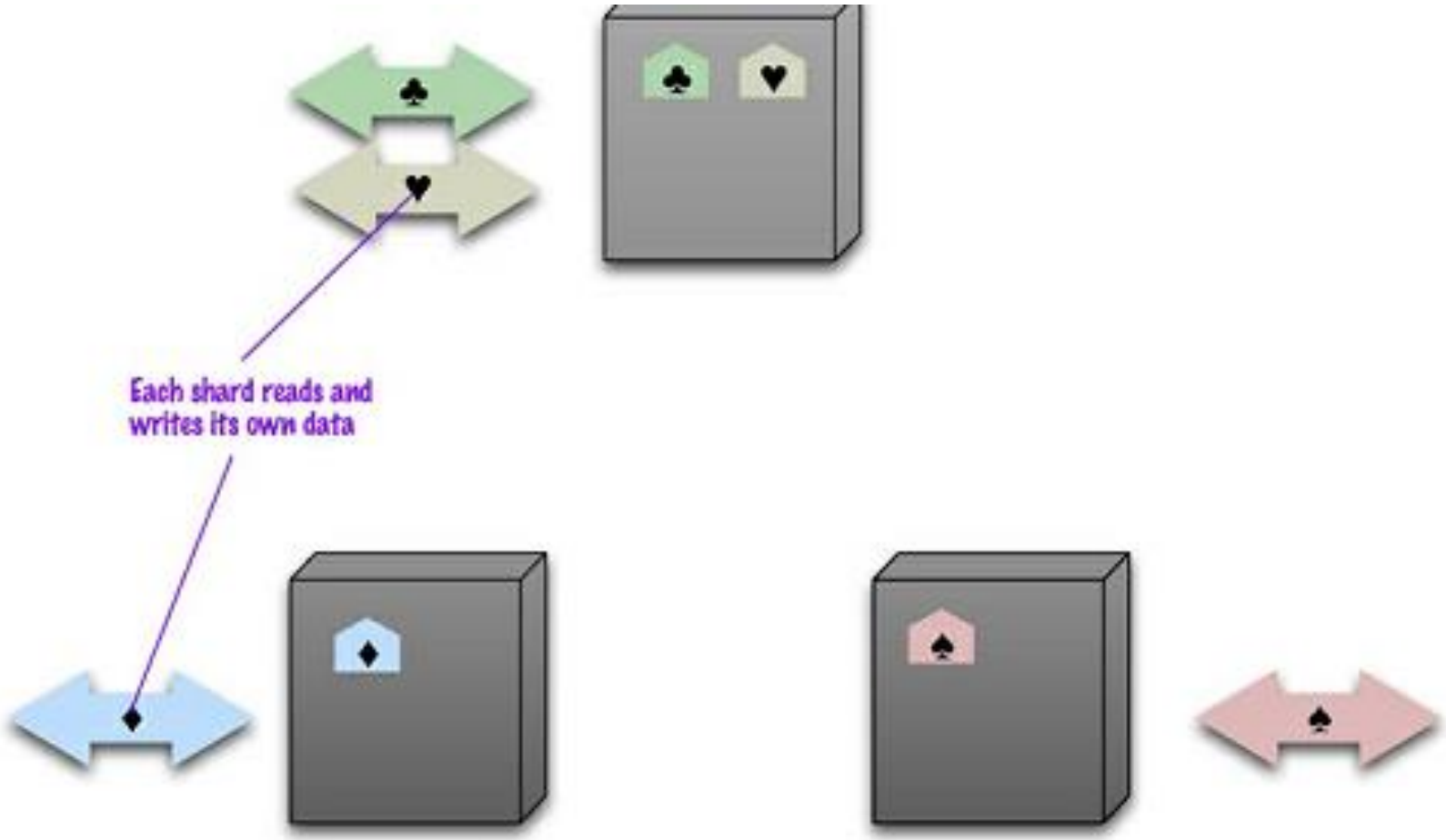
Quando vários usuários acessam um mesmo servidor este passa a não suportar, vindo a apresentar atrasos ou falhas;

Uma das possibilidades de solucionar esse problema é investir em escalabilidade horizontal:

- Fragmentação (**sharding**)

Cada usuário se comunica com um servidor e a carga é dividida.

Fragmentação



Fragmentação

Principal desafio:

- **Dividir a carga entre os vários nós**

Existem algumas saídas para tentar obter isso:

- Agrupar dados acessados em conjunto em um só nó
- Organizar dados dentro dos nós

Fragmentação

Agrupar os dados acessados em conjunto em um só nó se torna menos complexo com o uso de agregados de dados

Já para organizar os dados existem algumas alternativas:

- Organizar dados pela localização geográfica. Exemplo: servidores regionais com dados por região.
- Organizar dados pelo perfil de acesso. Exemplo: regras específicas para cada dia da semana, com base no padrão de acesso.

Alguns bancos oferecem serviço de auto fragmentação.

Replicação

Na replicação, os dados são copiados em vários nós

Existem duas formas principais:

- Replicação mestre/escravo
- Replicação ponto a ponto

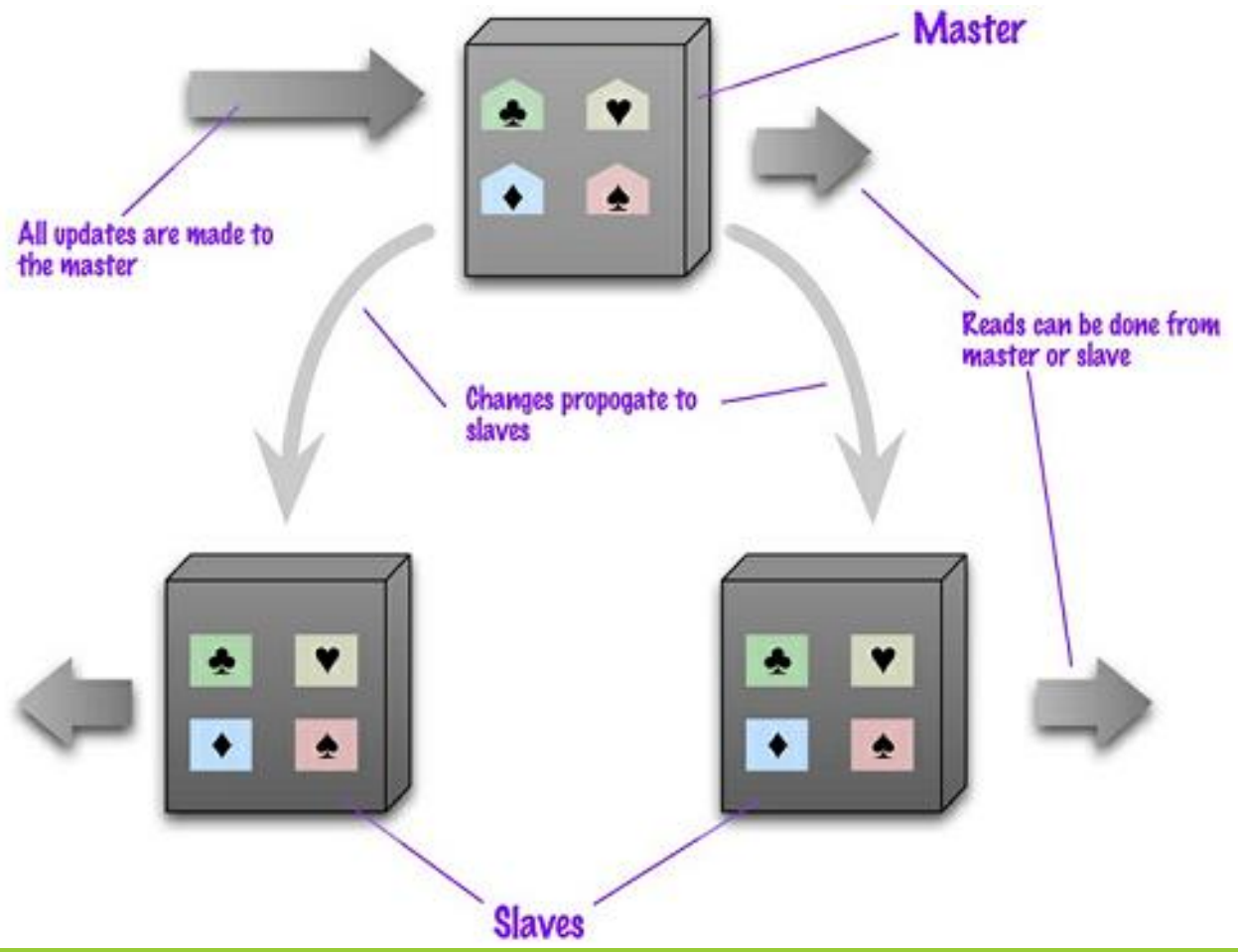
Cada uma possui algumas particularidades.

Replicação mestre/escravo

Existem dois tipos de nós:

- Nó mestre (primário): fonte oficial dos dados, responsável por realizar as modificações nos dados
- Nós escravos (secundários): são sincronizados com o mestre e permitem a realização de leituras nos dados

Replicação mestre/escravo



Replicação mestre/escravo

Oferece uma série de vantagens:

- Grande volume de consultas
- Facilmente escalável
- Resiliência de leitura
- Pode haver alternância de mestres

Replicação mestre/escravo

Por outro lado, existem algumas limitações:

- Atualizações só acontecem no mestre
- Abre margem para inconsistências
- Se o mestre falha antes de replicar uma atualização esta é perdida

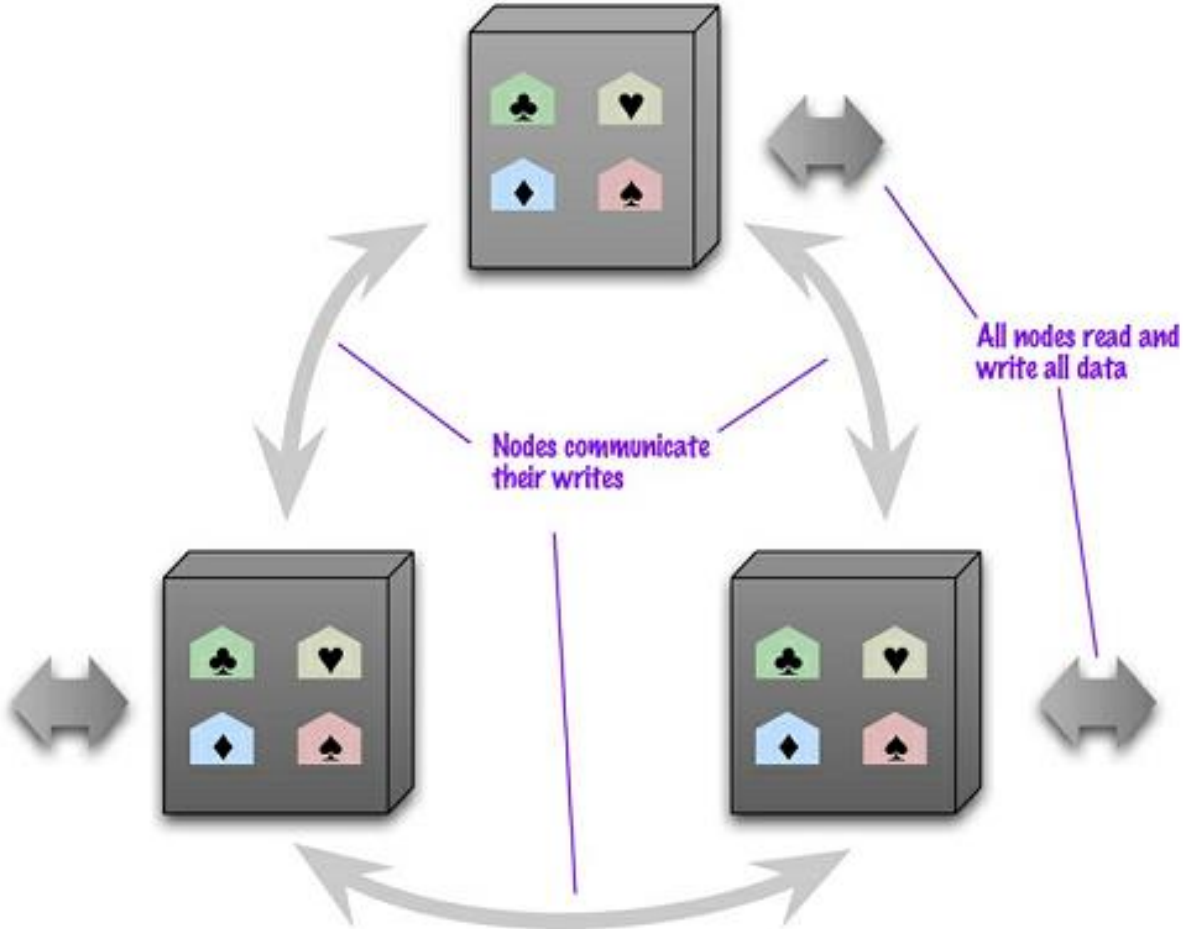
Replicação ponto a ponto (P2P)

Todos os nós possuem as mesmas responsabilidades:

- Realizam leituras e gravações

A falha de alguma das réplicas não impede a realização das operações.

Replicação ponto a ponto (P2P)



Replicação ponto a ponto (P2P)

Oferece algumas vantagens:

- Contorna melhor as falhas dos nós
- Facilmente escalável

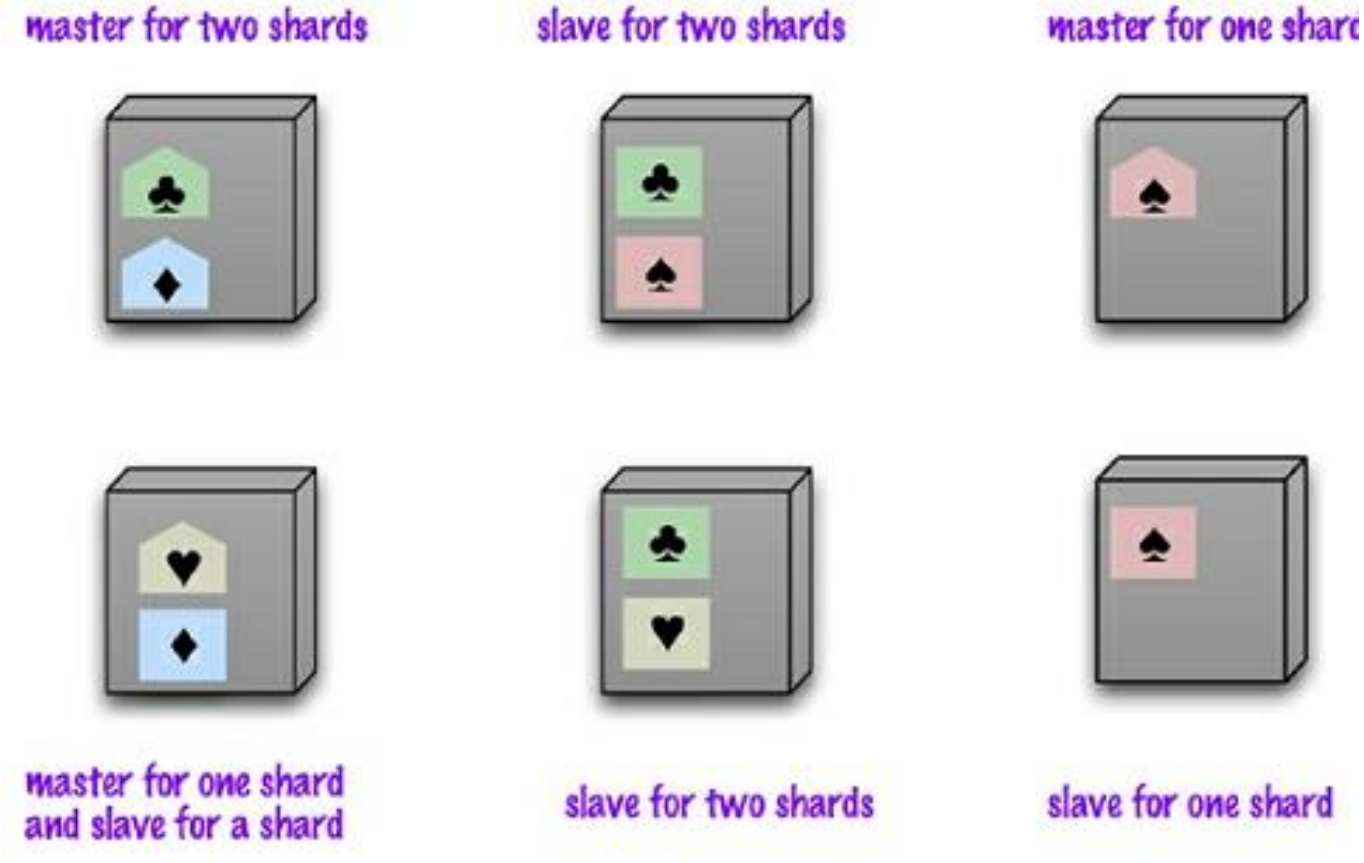
Replicação ponto a ponto (P2P)

Mas possui desvantagens:

- Abre margem para inconsistências
- Maior complexidade

Fragmentação e replicação

É possível combinar as estratégias



Consistência

Basicamente existem duas formas de consistência que devemos nos preocupar quando se trata de bancos distribuídos:

- Consistência de atualização
- Consistência de leitura

Consistência de atualização

Dado o seguinte cenário:

- Duas pessoas desejam reservar o último quarto de um determinado hotel ao mesmo tempo.

Consistência de atualização

Quando as solicitações chegam ao servidor elas são serializadas:

- A primeira que chegar será feita e depois a segunda...

Porém, existirá uma inconsistência:

- A atualização 2 foi baseada em um estado do banco anterior ao estado 1.

Consistência de atualização

Existem duas soluções:

- Solução pessimista: Não permite que ocorram os conflitos
- Solução otimista: Permite que o conflito ocorra, e soluciona-o

Consistência de atualização

Solução pessimista (conservador):

Pode utilizar um espécie de bloqueio: Quando um cliente vai realizar uma atualização ele obtém o controle daquele campo, e somente ele pode altera-lo.

Somente uma atualização será bem sucedida.

Consistência de atualização

Solução otimista (agressivo)

Pode ser utilizado uma espécie de controle de versão: antes de atualizar, o cliente verifica se houve alguma atualização desde a sua última leitura.

É mais complexo com múltiplos servidores, pois podem haver valores diferentes para o mesmo campo.

Consistência de leitura

A consistência de atualização não garante que quem leia os dados recebe informações consistentes.

Usuário A

Usuário B

No exemplo anterior, A e B estão reservando o último quarto do hotel. Um usuário C pode ver o quarto como disponível mesmo após reservado.

Consistência de leitura

A maioria dos bancos NoSQL assume que não é possível ser consistente o tempo todo em um banco distribuído.

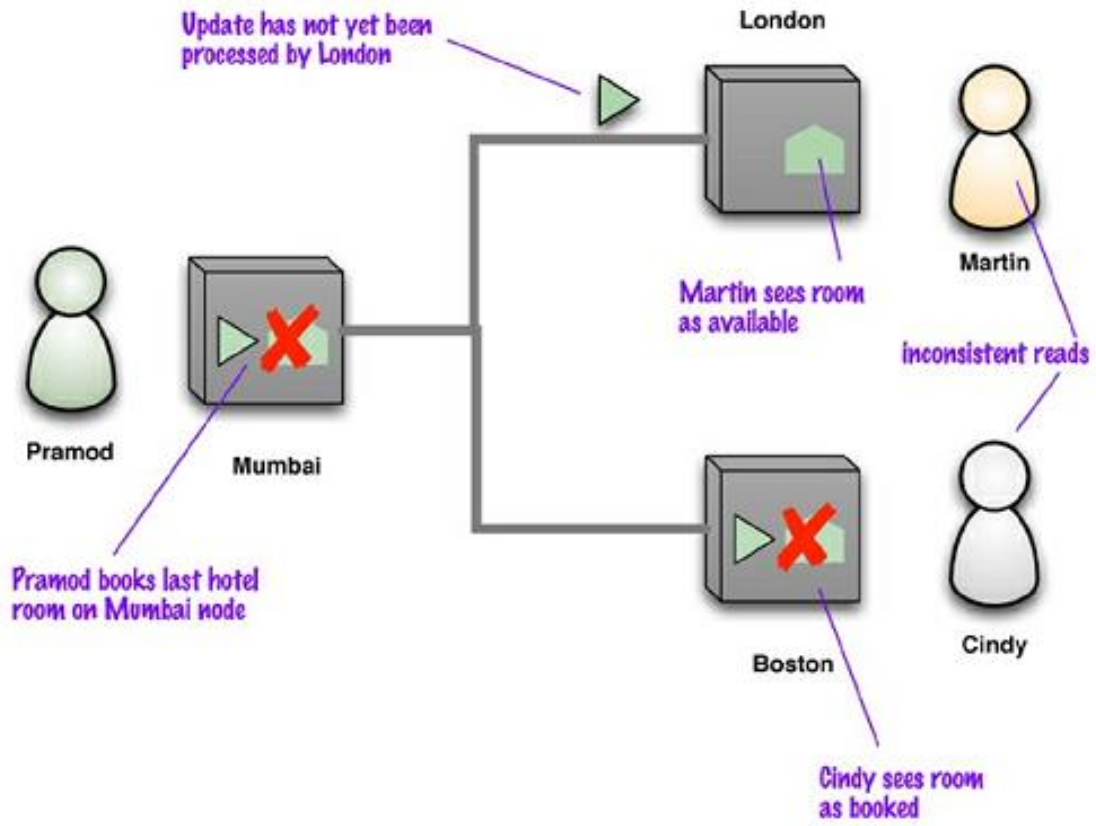
Eles trabalham com o conceito de **janela de inconsistência**:

- Tempo até o banco voltar a ser consistente
- O SimpleDB, por exemplo possui uma janela de inconsistência menor que 1 segundo.

Consistência de leitura

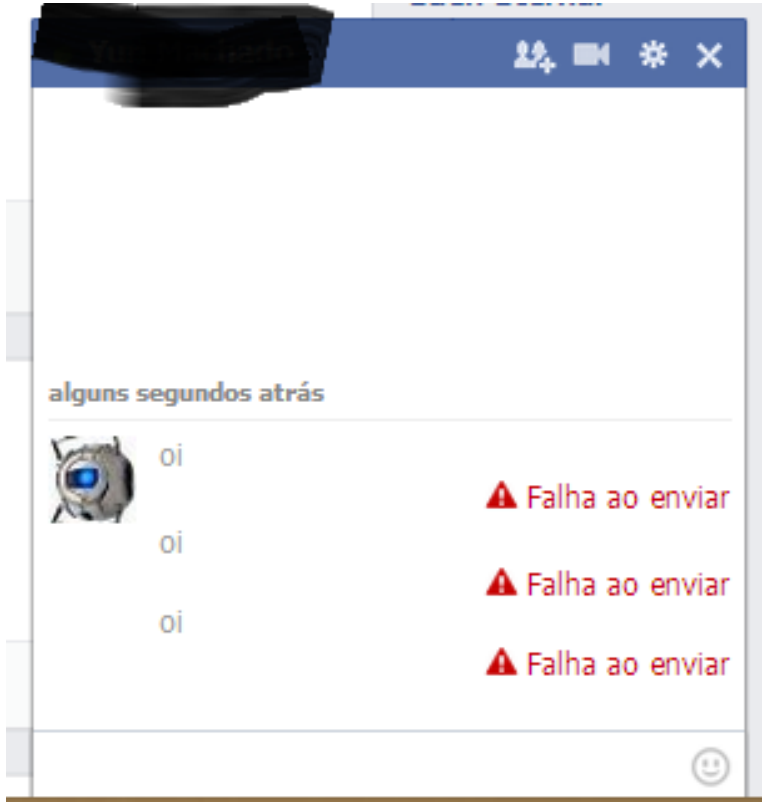
Quando utilizamos replicação, temos também o tempo para a informação ser replicada

- Questões de rede influenciam.



Consistência de leitura

Exemplo: em um chat, você envia a mensagem, mas devido ao atraso, a mensagem é identificada como não enviada.



Consistência de leitura

Dependendo da aplicação, a consistência pode não ser um requisito essencial.

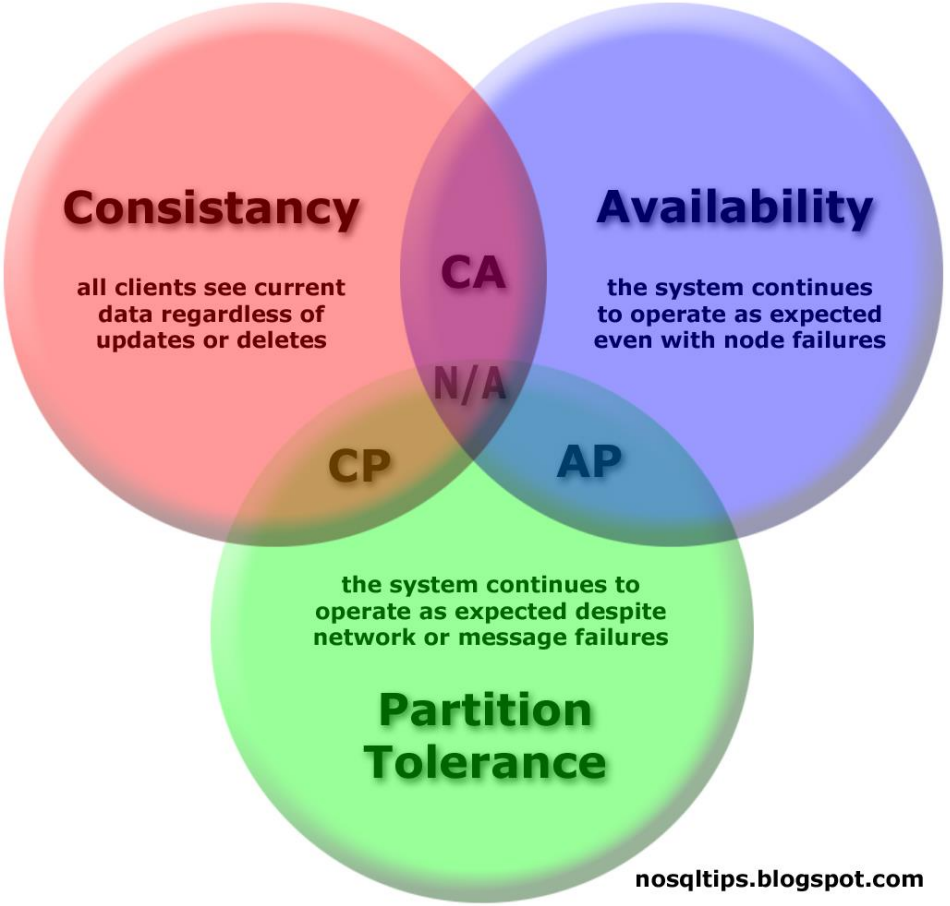
- Por exemplo, para um hotel é mais cômodo enviar um e-mail pedindo desculpas por um engano que perder uma reserva por problemas na rede.

Teorema CAP

O teorema CAP diz que das propriedades, só podemos garantir duas delas:

- **Consistência**
- **Disponibilidade**: Toda solicitação recebida por um nó sem falhas deve resultar em uma resposta.
- **Tolerância a partições**: O cluster pode suportar falhas que crie múltiplas partições incapazes de se comunicar entre si.

Teorema CAP


Consistancy
all clients see current data regardless of updates or deletes
Availability
the system continues to operate as expected even with node failures
CA
N/A
CP
AP
the system continues to operate as expected despite network or message failures
Partition Tolerance
nosqltips.blogspot.com

Teorema CAP

Se usarmos um único servidor ele é CA

É possível ter um cluster CA, mas se ele partir todo o sistema cai.

Na prática o teorema CAP implica que devemos relaxar a consistência para garantir disponibilidade.

Teorema CAP

Os negócios atuais toleram falta de consistência:

- Carrinho de compras
- Reservas de hotéis

BASE

Os bancos NoSQL seguem o BASE ao invés do ACID:

- Basically Available: Basicamente disponível
- Soft State: Estado leve
- Eventually consistent: Eventualmente disponível

- Uma aplicação funciona basicamente o tempo todo (basicamente disponível), não tem de ser consistente todo o tempo (estado leve) e o sistema torna-se consistente no momento certo (eventualmente consistente).