INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

LICENCIATURA EM ENGENHARIA DE INFORMÁTICA E COMPUTADORES

SEMESTRE DE VERÃO 2011/2012

**Sistemas Distribuídos**

*Optimistic Data Replication*

***Grupo:***

* G05D

***Alunos:***

* João Pires 31933
* José Casimiro 32713
* António Dente 33168

Índice

[Introdução 3](#_Toc328073832)

[Replicação Pessimista 3](#_Toc328073833)

[Replicação Otimista 3](#_Toc328073834)

[Escolhas de desenho 4](#_Toc328073835)

[Complexidade e Eficiência 4](#_Toc328073836)

[Concorrência 5](#_Toc328073837)

[Eficiência 5](#_Toc328073838)

[Qualidade 5](#_Toc328073839)

# Introdução

Em quase todos os sistemas distribuídos, existe algum tipo de *data* ou informação que é necessário partilhar por vários servidores. Estejam estes armazenados em base de dados ou em memória volátil é imperativo que se mantenham consistentes e coerentes independentemente do local onde são acedidos.

Aqui entra a replicação de dados. Mantendo múltiplas cópias de informação em servidores separados. Deste modo pode-se aceder a um servidor mais próximo de nós mesmo que a informação tenha originado do outro lado do globo, melhorando a velocidade de receção e facilitando a alteração de informação, caso esta funcionalidade seja permitida.

Existem duas técnicas base de replicação de dados, replicação pessimista e otimista. Neste relatório iremos dar uma pequena explicação da replicação pessimista embora nos vamos focar na otimista.

# Replicação Pessimista

Numa replicação pessimista (tradicional) tenta-se criar a ilusão de que apenas uma cópia dos dados existe, por exemplo, apenas existe uma base de dados. Para criar esta ilusão não se permite acesso a qualquer informação que não seja provada como a mais atual. Para tal recorre-se a técnicas de *multi locks* e sempre que é realizada uma alteração a um dado esta é replicada para todas as cópias. Daí sere chamada de pessimista.

Esta abordagem, de facto, garante uma grande consistência nos dados mas é irrealista num cenário *web*. Embora em cenários *lan* seja possível a sua concretização, tendo em conta baixos valores de latência e pouca probabilidade de falha mecânica, num cenário *web* estes pressupostos não podem ser garantidos.

# Replicação Otimista

As técnicas de replicação otimista, em oposição à pessimista, pressupõem que, qualquer erro ou falha, irá ocorrer poucas vezes, se alguma.

Com base neste pressuposto, uma técnica otimista, permite que os dados sejam acedidos mesmo que não sejam comprovados como os mais atuais. Isto permite um ambiente de trabalho muito mais fluido e “isolado”. Deste modo podemos garantir uma melhor disponibilidade de informação ao custo do ocasional conflito, visto um dado poder ser alterado por duas pessoas diferentes em duas réplicas distintas.

Após uma alteração, ao contrário de numa replicação pessimista, a informação da alteração é partilhada em *background* e caso ocorra um conflito este pode ser resolvido automaticamente ou com ajuda do utilizador.

Qualquer sistema de replicação deve ter o conceito de unidade de replicação mínima, a esta unidade é dado o nome de Objeto. Um Objeto pode ter réplicas por vários locais e um local armazenar réplicas de vários objetos. Com estes objetos é que são partilhadas as alterações realizadas em cada local. Embora a alteração possa ser realizada em apenas uma máquina (single-master / writer) ou em várias (multi-master / writer).

Qualquer sistema que utilize replicação otimista deve também permitir acesso a dados que estejam “desligados” das outras réplicas. Podemos pensar num sistema móvel em que o utilizador se conecta á internet apenas quando necessário. Deste modo o utilizador pode realizar as alterações em modo *offline* e depois realizar a propagação das alterações quando se ligar à internet. Deste modo podem ser armazenadas as operações que foram realizadas sobre um objeto ou apenas o objeto em si, estes dois modos de alteração de objetos / propagação de alterações têm algumas diferenças e iram ser discutidos mais á frente.

# Escolhas de desenho

Quando estamos a desenhar uma aplicação que contenha replicação de dados devemos pensar nalguns temas que nos permitem visualizar a complexidade do nosso sistema e como o vamos implementar.

## Complexidade e Eficiência

Um dos pontos que devemos analisar é a quantidade de escritores que existe, que réplicas, se alguma, permitem a alteração de dados. Este ponto ajuda a definir a complexidade base do nosso sistema. Quanto mais escritores, maior a possibilidade de conflitos e mais frequente devem ser os *updates* de informação. Num sistema com apenas um escritor, como por exemplo no DNS, não há necessidade de tratamento de conflitos visto que estes não irão ocorrer, basta que, quando uma alteração for realizada, as réplicas sejam informadas dessa alteração. Num sistema com múltiplos escritores é necessário um tratamento de conflitos pois mais do que uma réplica pode ter realizado alguma alteração sobre um objeto.

Outro ponto que afeta diretamente a complexidade do sistema são as operações realizadas, se a alteração de um objeto é informada pelo estado do objeto ou apenas pela operação realizada sobre este. No primeiro caso irão ocorrer mais conflitos que no segundo pois, com o estado do objeto torna-se mais simples indicar apenas que houve uma alteração mas pode ser complicado saber onde, ao saber a operação sabemos exatamente o que foi alterado e pelo que foi alterado. Esta última solução permite um melhor tratamento de conflitos, seja ao verificar que a alteração ocorreu em locais distintos e portanto não “chocam”, ou mesmo a aplicar melhorias na própria ordem em que as operações foram realizadas, embora para esta última situação seja necessário que o sistema conheça a sua semântica.

## Concorrência

Também tem de ser pensado como é realizado o scheduling, ou seja, como são ordenadas as operações quando é realizada alguma alteração sobre um objecto. Este scheduling pode ser também sintático ou semântico. Num scheduling sintático apenas é tomado em conta onde, quando, e por quem foi realizada a operação, ou seja, se três operações A, B e C forem realizadas numa máquina, se as três operações fossem realizadas em máquinas distintas mas pela mesma ordem, a ordem pela qual elas iriam ser realizadas nas réplicas seria a mesma. Ao ter um scheduling semântico é tido em conta informações do sistema para melhor ordenar as operações. Por exemplo, se no caso anterior, a realização da operação C antes de B causasse que todas tivessem um resultado positivo enquanto que pela sua ordem original uma delas obtivesse um resultado negativo, se não houver inconvenientes para o sistema, este pode reordenar essas duas operações.

No campo de concorrência temos também o tratamento de conflitos. Como já disse anteriormente, os conflitos podem ser tratados automaticamente ou com input do utilizador. Podemos pensar no caso do SVN ou do GIT em que é realizado um *merge* automático de ficheiros mas em caso de conflitos que o sistema não consiga resolver, é pedido ao utilizador que os resolva e volte a enviar o resultado. Neste caso temos uma combinação das duas soluções.

## Eficiência

Em aspetos de eficiência é necessário pensar na estratégia da propagação, como são propagadas as operações entre locais. Também aqui temos várias soluções, podem ser realizados *pull’s* manuais (por exemplo em PDA’s) ou periódicos (como no caso do DNS) ou por *push*, através de um update periódico. Quanto mais curto o período entre atualizações menor a inconsistência de informação e número de conflitos, mas também aumenta a complexidade do sistema e o overhead.

## Qualidade

Aqui entra em conta as garantias que o sistema providencia. Se é garantido apenas o estado final do objeto, podendo o estado intermédio (entre operações) ser um pouco inconsistente, ou se todos os estados (intermédios e final) são garantidos. Aqui entra também em conta o scheduling pois como tinha informado é possível que as operações sejam realizadas por ordens diferentes da realizada pelo utilizador desde que o resultado final seja o mesmo.