# Deteção de relações de concorrência e “Happens-Before”

Um sistema de replicação otimista suporta várias operações em simultâneo através do escalonamento das mesmas e da deteção de conflitos entre elas. Para ser possível o escalonamento o sistema tem de saber quais os eventos e a ordem pela qual ocorreram.

Contudo num ambiente distribuído onde o atraso nas comunicações é imprevisível não é possível ordenar de forma absoluta as ocorrências dos eventos.

Para resolver este problema foi proposto por Lamport [1] num artigo de 1978 o conceito de *“happens-before”* que consiste em capturar relações de precedência entre eventos e tentar definir entre eles a ordem de ocorrência. Considere-se duas operações A e B efetuadas sobre os processos pA e pB respetivamente. A operação A *“happens-before”* B quando:

- pA = pB e a operação A for submetida antes de B;

- pA ≠pB e a operação B for submetida após pB receber e executar a operação A;

- Para uma dada operação C, A *“happens-before”* C e C *“happens-before”* B;

No caso de não ser detetada a relação de *“happens-before”* as operações dizem-se concorrentes. Este tipo de relações é usado em vários cenários na replicação otimística como o de ordenação de operações, deteção de conflitos e propagação de operações.

## Algoritmos de deteção e representação de relações *“happens-before”*

Existem vários algoritmos para representar e detetar estas relações:

* **Representação Explicita** - Nome das operações precedentes é anexado à operação x;
* **Relógios vetoriais** - Cada processo contem um vetor de relógios onde para cada i é guardado o número de eventos ocorridos no processo Pi. Quando é submetida uma nova operação y no processo Pi este incrementa o seu valor de eventos ocorridos no vetor e anexa o novo vetor à operação y para atualizar os vetores dos outros processos;
* **Relógios lógicos e de tempo-real** 
  + **Lógicos** - Cada processo tem um relógio interno e ao ser feita uma operação x esse relógio é incrementado e anexado à operação. O processo destinatário da operação incrementa o seu relógio para um valor maior que o seu ou maior que o anexado na operação;
  + **Tempo-Real** - Comparação dos relógios físicos.
* **Relógios plausíveis** - Combinação de alguns aspetos dos relógios lógicos com vetoriais;

# Controlo de concorrência e consistência eventual

Informalmente consistência eventual significa que todas as réplicas irão eventualmente estar sincronizadas quando o sistema se encontra inativo por dado período de tempo. Um objeto replicado está eventualmente consistente quando respeita as seguintes condições assumindo que todas as réplicas partilham o mesmo estado inicial:

* Em qualquer momento, para cada réplica existe um prefixo do *“schedule”* o qual é equivalente ao prefixo do *“schedule”* das restantes réplicas. A esta condição dá-se o nome de *“committed prefix”*.
* O *“committed prefix”* de cada réplica cresce de forma monotónica ao longo do tempo.
* Todas as operações não abortadas no *“committed prefix”* satisfazem as suas pré-condições.

Para atingir a consistência eventual o sistema tem de adotar uma política de scheduling, detetar conflitos e posteriormente resolvê-los.

## Schedulling

Existem essencialmente duas políticas de *“scheduling”*:

* **Sintático** - Baseado em informação de caracter geral como quando, onde e por quem foram submetidas operações
* **Semântica** - Baseado nas relações semânticas entre as operações tais como a comutatividade, onde duas operações consecutivas se forem comutativas a sua ordem de execução pode ser trocada mesmo que haja uma relação de *"happens-before".*

Existem também outros tipos de *“scheduling”* semântico tais como ordenação canónica, transformação operacional e abordagem de otimização.

## Deteção de conflitos

Uma operação A está em conflito quando a sua pré-condição não é satisfeita, dado o estado da réplica após várias tentativas e de aplicar operação antes de A no *“schedule”* corrente. A gestão de conflitos envolve duas fases, deteção de conflitos e resolução dos mesmos.

Tal como nas políticas de *“scheduling” e*xistem duas abordagens para deteção de conflitos em sistemas otimistas:

* **Abordagem sintática** - Duas operações estão em conflito quando são concorrentes não havendo relação *“happens-before”*.
* **Abordagem semântica** - Duas operações estão em conflito quando ambas têm a mesma semântica. Por exemplo a escrita concorrente de dois ficheiros distintos no mesmo sistema de ficheiros não é um conflito mas a alteração concorrente do mesmo ficheiro é.

## Resolução de conflitos

A resolução de conflitos tem como finalidade reescrever ou abortar operações que potenciem conflitos.

A resolução de conflitos pode ser manual ou automática:

**Manual** - O sistema exclui a operação conflituosa do *“schedule”* e apresenta duas versões do objeto. Cabe ao utilizador criar uma versão nova com *“merging”* das duas versões e voltar a submeter a operação. (**e.g. – Sistema de controlo de versões Git**).

**Automático** – É chamado um procedimento especifico da aplicação que pega nas duas de um objeto e cria um novo. Por exemplo, atualizações concorrentes a um ficheiro de uma diretoria de e-mail podem ser resolvidas através da computação da união das mensagens de duas réplicas.

## Protocolos de compromisso

Para implementar as 3 etapas da consistência eventual existem os protocolos de compromisso onde a permanência das operações é acordada entre os intervenientes nomeadamente a política de *“scheduling”* e a técnica de resolução de conflitos.

Um dos protocolos é o de compromisso implícito por conhecimento comum. Os sistemas que implementam este protocolo fazem uso de timestamps na ordenação das operações de forma determinística e os conflitos ou não existem ou são ignorados.

Existem ainda outros protocolos como o de acordo em segundo plano e por consenso.

Referências

[1] - [Time, Clocks and the Ordering of Events in a Distributed System](http://research.microsoft.com/en-us/um/people/lamport/pubs/time-clocks.pdf)