Processamento de erros

Detectar / Recuperar/ Mascarar

1º passo: Detecção.

A detecção de um erro pode ser realizada por mecanismos como:

- códigos de detecção de erros,
- "timeouts"
- "watchdogs"
- verificações sintácticas ou semânticas
- comparações bit-a-bit por hardware
- ...

- Uma vez detectado o erro, este deve ser confinado para que não se propague.

Se não existir redundância suficiente para recuperar o sistema, este deverá ser desligado (caso seja possível). Por exemplo, um sistema de gestão de tráfego ferroviário pode atingir um estado estável parando todos os comboios, mas um avião não pode parar antes de aterrar).

2º passo: Recuperação

Se for possível, o erro deverá ser recuperado. Existem duas formas de recuperação:

-"backward recovey" – recuperação para trás

Consiste em regressar a um estado anterior, em que o estado do sistema estava correcto, e recomeçar a partir daí.

[No limite☺: "sair e voltar e entrar" .... "desligar e ligar de novo"]

Pode ser, por exemplo:

- se um erro foi detectado numa mensagem então é pedido o seu reenvio

- se, ao executar uma asserção, se verifica que um valor está incorrecto então é executado um algoritmo alternativo (Recovery Blocks).

Na maioria dos casos, recuar para um estado correcto não é tão simples como possa parecer:

- o erro pode já ter sido propagado a outros componentes

- recuar significa desfazer computações intermédias que podem por sua vez ter desencadeado outras

- eventualmente podem ter sido produzidos efeitos fora do sistema e não ser possível o sistema reverter essas acções

A forma mais comum de "backward error recover" é o estado do sistema ser guardado de tempos a tempos (usando um intervalo de tempo pré-estabelecido ou em pontos da computação determinados à partida)

Ao estado do sistema que é armazenado, chamamos

"checkpointing" - "salvaguarda de estado"

Quando ocorre um erro, a computação recorre ao último "checkpointing".

Segunda forma de recuperação de erros:

"forward recovery" - recuperação para a frente

Consiste em tomar medidas que cancelem ou diminuam o efeito do erro.

Por vezes pode não haver tempo para regressar atrás e recuperar a partir de um estado correcto.

Por exemplo,

Se a leitura de um sensor se perder é preferível esperar pela próxima leitura do que pedir a retransmissão do sinal anterior (este pode já estar desactualizado).

3º passo: Mascarar o erro

Se o sistema tem redundância suficiente para que o serviço correcto seja executado sem que o utilizador se aperceba, então o erro é mascarado.

Hardware-based fault tolerance Versus Software-based fault tolerance

Técnicas de tolerância a falhas baseadas na **replicação de hardware** (como por exemplo a redundância modular tripla) têm algumas desvantagens:

- Não dão resposta a falhas que afectem todas as réplicas, como uma catástrofe (inundação, incêndio) nem a erros de desenho.

- Hardware especializado é caro e mais difícil de actualizar do que por exemplo componentes do tipo COTS (Commercial Off-The-Shelf).

- Falhas de hardware são apenas uma parte das falhas que podem afectar um sistema. Com a crescente complexidade do software poderão ser uma fracção cada vez menor.

A evolução é no sentido da replicação de componentes de software (começando por reproduzir as abordagens de replicação do hardware)

Software-based fault tolerance

- É mais simples adicionar ou remover uma réplica de software do que uma réplica de hardware.

- Réplicas de software podem ter diferentes granularidades, desde replicar uma Base de Dados a replicar um simples função.

Réplicas de software podem ser executadas no mesmo hardware ou serem distribuídas, levando à

Tolerância a Falhas Distribuída

Cada réplica de uma dada computação pode estar localizada num outro nó de um sistema distribuído.

Tolerância a Falhas Distribuída

Torna possível detectar:

- falhas de hardware
- falhas transitórias de software ("Heisenbugs")
- desastres

- Falhas transitórias de software tornam-se mais fáceis de detectar em resultado da inerente heterogeneidade de um sistema distribuído (diferentes arquitecturas, sistemas operativos, ...,

-Outras falhas de desenho de software serão difíceis de detectar e menos que cada réplica seja criada por diferentes programadores (N-version programming).

Para que a tolerância a falhas distribuída funcione será necessário ter uma rede de comunicação tolerante a falhas !!

Protocolos de comunicação tolerantes a Falhas

Num canal de comunicação pode haver perda, alteração da ordem ou corrupção de mensagens devido a ruído, atenuação do sinal, "overflow" do "buffer" do receptor, ou de "router" intermédio, etc,

Há que garantir que uma sequência de mensagens m1, m2, ..., m_n enviada do emissor para o receptor deve:

- Chegar ao receptor não corrompida
- Chegar pela mesma ordem porque foi enviada

=> Para garantir que a mensagem chega é, antes de mais, necessário garantir que existe mais do que um percurso alternativo entre cada dois nós.

Protocolos de comunicação tolerantes a Falhas

=> Para detectar a corrupção de uma mensagem, são usados códigos de detecção de erros:

- bits de paridade, CRC's, etc

- Se a mensagem foi corrompida, é rejeitada.

=> Para garantir a entrega e a ordem:

- Protocolos de entrega fiável de mensagens:

. RDT (Reliable Data Transfer)

. Sliding-Windows

...

Detecção de avarias

O sistema, como um todo, não pode ser tolerante às suas próprias avarias, isto é, quando o sistema tem a avaria, já nada pode ser feito.

Um sistema pode ser tolerante às avarias dos seus componentes.

O objectivo da tolerância a falhas é

evitar a avaria do sistema completo, quando ocorre uma falha nalgum dos seus subsistemas.

Ou seja, mascarar a avaria de algum dos seus subsistemas.

Para detectarmos a avaria de um componente, o alvo, precisamos de outro componente, o detector. É necessário ainda um canal de comunicação entre os dois.

Detecção de avarias

Resumindo, para detectar a avaria de um componente é necessário adicionar dois componentes ao sistema que por sua vez também podem falhar.

Avarias no detector:

Se o detector avaria, um dos possíveis outputs, é ser assinalada a avaria do componente alvo apesar do mesmo estar correcto:

- Falso Alarme

O detector deve ser construído de tal forma que exiba uma fiabilidade muito maior que a do componente alvo:

Deve ser mais simples, ou ser construído com um hardware mais fiável, ou ambos.

Detecção de avarias

O sistema deve ser construído de forma a que as consequências de uma detecção errada sejam menos gravosas do que a não detecção da avaria.

(Ex. – Sistema de detecção de colisões)

Avarias no canal de comunicação

Se houver perda ou atraso das mensagens pode ser difícil, ou impossível, distinguir a avaria de um componente do funcionamento incorrecto do canal.

Detecção local de avarias

Isto é, detecção de avarias quando o alvo e o detector estão suficientemente próximos para que a comunicação seja fiável **(***mesma máquina ou mesma "board")*

Detecção local de avarias

Exemplos:

- rotinas para verificação de códigos de erros

-“watchdog timers”

Testam se uma dada computação progride a um certo ritmo.

Implementação em software: Um processo (alvo) tem de periodicamente atribuir um valor a uma posição de memória. Outro processo (o detector) verifica essa posição de memória e “apaga” o valor. Sempre que encontra o valor não modificado produz um sinal de avaria

Para que este esquema funcione, é necessário que o processo detector tenha prioridade adequada e acesso à informação necessária para poder distinguir entre uma verdadeira avaria e uma sobrecarga do sistema.

Diagnóstico do sistema

Podemos garantir o modelo anterior (um alvo, um detector) considerando que todos os componentes do sistemas são simultaneamente alvos e detectores.

Cada componente pode prestar um serviço e testar outros componentes.

Fazer o diagnóstico de um sistema consiste em

identificar que componentes estão avariados

com base nos resultados obtidos quando os vários componentes se testam mutuamente.

Assumimos que

- o resultado dos testes feitos por componentes correctos são confiáveis.

Diagnóstico do sistema

- Componentes avariados podem assinalar,

. componentes correctos como avariados

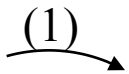
. e componentes avariados como correctos

Seja,

 - componente correcto

 - componente avariado

 - liga o componente detector ao componente alvo

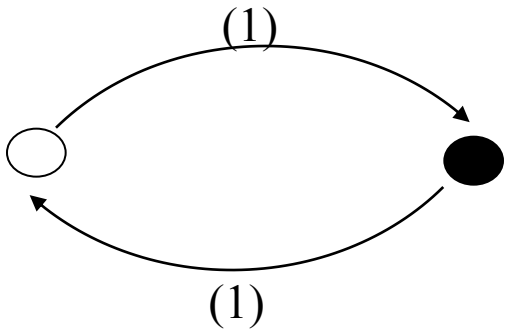


- indica que o alvo é assinalado como avariado

-indica que o alvo é assinalado como correcto

Diagnóstico do sistema

Suponhamos a situação:



Ambos os detectores assinalam o alvo como avariado.

O que podemos concluir?

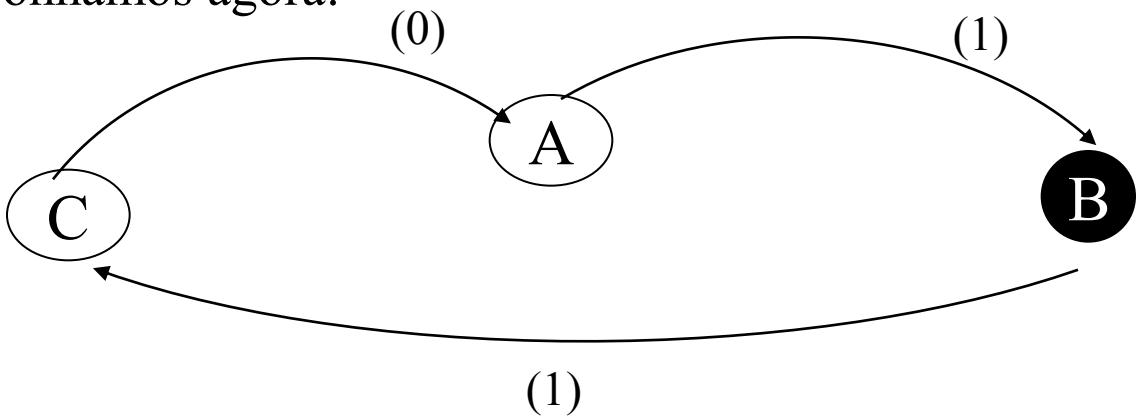
Demonstra-se que num sistema com f componentes avariados são necessários

n>= 2 f + 1 processos para diagnosticar a avaria e

cada componente terá de ser testado por pelo menos f outros componentes.

Diagnóstico do sistema

Suponhamos agora:



Para identificar um nó avariado este deverá ter:

i) um arco convergente que assinala avaria

ii) o nó origem desse arco deve ser assinalado como correcto

B verifica as duas condições, logo está avariado

C é considerado correcto, apesar de ser assinalado como avariado (não verifica ii)).

Para efectuar o diagnóstico, os resultados de cada verificação devem ser analisados por um componente centralizado, geralmente exterior ao sistema.

Detecção distribuída de avarias

Mais difícil que a detecção local, porque a comunicação entre o alvo e o detector também pode sofrer avarias.

Comecemos por considerar que:

- queremos detectar a avaria de processos

- que os processos quando falham deixam de funcionar (crash)

- que o sistema é síncrono (isto é, existem limites de tempo para a computação e comunicação)

Neste caso um processo está correcto se dá evidências de estar em funcionamento, através da emissão de mensagens.

Periodicamente, o processo envia uma mensagem,

"heartbeats" ou "I'm alive messages"

Heartbeats são enviados espontaneamente pelo alvo,

em alternativa o detector envia uma mensagem de teste e o alvo responde com uma mensagem do tipo "I'm alive"

Se o processo realiza tarefas em que interactua com outros processos através de mensagens, essas mensagens são usadas para monitorizar a sua actividade.

Num sistema em que todos os processos possam comunicar entre si (full connectivity) ,

cada processo pode ser simultaneamente alvo e detector, deveremos ter um sistema em que

a detecção de avarias seja consistente, isto é:

Se um processo avaria, ele deverá ser considerado em falha por todos os processos correctos no sistema.

Esta situação é possível se tivermos um canal de comunicação sem erros.

Se um processo falha, todos os outros vão notar a ausência das suas mensagens (hearbeats) e detectar a avaria. Neste caso dizemos que temos um detector de avarias perfeito

O que acontece se houver erros na comunicação?

Se o erro de comunicação for fácil de tratar há que assegurar a fiabilidade da comunicação, por exemplo a perda de um número **limitado** de mensagens, é tratada com a replicação do envio das mensagens.

Detectores perfeitos, poucas vezes são possíveis de implementar.

Dois problemas principais:

Não há limites para o tipo e número de falhas que podem ocorrer num canal de comunicação.

A maioria dos sistemas são assíncronos (isto é, não há limites para o tempo de execução nem para o tempo de comunicação)

Comunicação Tolerante a Falhas

Como assegurar que dois ou mais processos trocam informação na presença de erros no canal de comunicação ou em algum dos processos.

Tipos de avarias possíveis:

mensagens fora de tempo (timing failures)

omissão de mensagens (omission failures)

corrupção de mensagens (message corruption)

Entrega fiável

Duas abordagens: mascaramento do erro ou recuperação do erro

Mascaramento do erro pode ser feito através de redundância espacial ou redundância temporal.

Entrega fiável de mensagens

Mascaramento do erro através de redundância espacial

Implementando várias ligações entre os processos em comunicação, a mensagem é enviada em simultâneo por todas as ligações.

Com k+1 ligações consegue-se tolerar k omissões no envio de uma mensagem.

Mascaramento do erro através de redundância temporal consiste em enviar a mesma mensagem várias vezes.

Em ambos os casos, o processo receptor terá que eliminar as mensagens duplicadas.

Entrega fiável de mensagens

Detecção e recuperação de erros

Baseada em confirmações (acknowledges) e tempos de espera (timeouts).

Um aknowledge pode ser enviado sempre que uma mensagem é recebida (positive acknowledge) ou apenas quando a perda de uma mensagem é detectada pelo receptor negative acknowledge).

Na confirmação positiva, a mensagem é reenviada se a confirmação (ack) não chega ao emissor dentro do timeout definido.

Na confirmação negativa a retransmissão é pedida pelo receptor, enviando um negative acknowledge (nack) ao emissor.

Entrega fiável de mensagens

Confirmação positiva

=> detecção de avarias mais rápida

Confirmação negativa

⇒ minimiza o tráfego mas requer que as mensagens sejam numeradas de forma a ser detectada a omissão de alguma.

Corrupção de mensagens

A maioria da falhas de valor pode ser detectada por códigos de erro.

Mas, a corrupção da mensagem pode ocorrer antes desta ser codificada. Falhas deste tipo podem ser toleradas através de redundância espacial, isto é comparando os valores produzidos para a mesma mensagem por diferentes componentes.