TOLERÂNCIA A FALTAS DISTRIBUÍDA

2005-2006
13 de Janeiro de 2006

Nota prévia

- O exame tem duração de duas horas e trinta minutos.
- Não se esqueça de identificar *todas* as folhas com o seu nome e número.

Abstracções Básicas

A decomposição de um problema complexo em sub-problemas é uma forma de facilitar a análise desse problema. Neste contexto, no desenvolvimento de algoritmos para sistemas distribuídos tolerantes a faltas encapsulam-se os protocolos de comunicação de baixo nível numa abstractção designada por "canais ponto-a-ponto perfeitos".

Questão 1 *(*1.5 valores*) Apresente uma definição deste tipo de canais.*

Questão 2 *(*1.5 valores*) Num sistema em que é possivel detectar falhas de processos com exactidão, qual o protocolo real que mais se aproxima desta abstracção?*

Questão 3 *(*1.5 valores*) Se não for possível detectar a falha de processos com exactidão, é possível concretizar esta abstracção? Esboçe a solução.*

Difusão fiável

Considere o seguinte algoritmo que pretende oferecer difusão fiável uniforme usando um detector de falhas perfeito:

Implements:
UniformReliableBroadcast (urb).

Uses:
BestEffortBroadcast (beb).
PerfectFailureDetector ($\mathcal{P}$).

function canDeliver(m) **returns** boolean **is**
return (m $\notin$ delivered);

upon event $\langle$ *Init* $\rangle$ **do**
delivered := pending := $\emptyset$;
correct := Π;
forall m **do** $\text{ack}_m := \emptyset$;

upon event $\langle$ *urbBroadcast*, m $\rangle$ **do**
pending := pending $\cup$ {(self, m)};
trigger $\langle$ *bebBroadcast*, [DATA, self, m] $\rangle$;

upon event $\langle$ *bebDeliver*, p_i, [DATA, s_m, m] $\rangle$ **do**
$\text{ack}_m := \text{ack}_m \cup \{p_i\}$;
if $((s_m, \text{m}) \notin \text{pending})$ **do**
pending := pending $\cup$ $\{(s_m, \text{m})\}$;
trigger $\langle$ *bebBroadcast*, [DATA, s_m, m] $\rangle$;

upon event $\langle$ *crash*, p_i $\rangle$ **do**
correct := correct $\setminus \{p_i\}$;

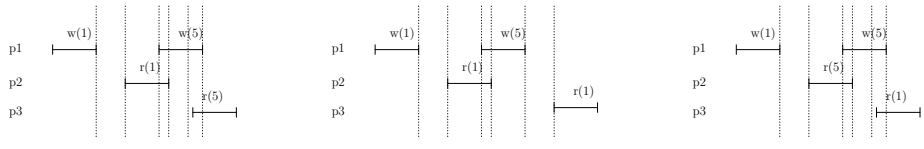
upon exists $(s_m, m) \in$ pending **such that** canDeliver(m) $\wedge$ $m \notin$ delivered **do**
delivered := delivered $\cup$ $\{m\}$;
trigger $\langle$ *urbDeliver*, s_m, m $\rangle$;

Questão 4 *(*1 valor*) O algoritmo possui um erro. Corrija-o.*

Questão 5 *(*1.5 valores*) Altere o algoritmo para não necessitar de recorrer a um detector de falhas perfeito. Indique que hipóteses adicionais é necessário fazer acerca do sistema para este problema ter solução.*

Registos

Considere as seguintes execuções de acesso a um registo:



(a) (b) (c)

Figura 1: Execuções de registos

Questão 6 *(*1 valor*) Apenas uma das execuções é válida para registos atómicos, outra só válida para registos regulares (mas não atómicos) e outra não é válida nem em registos atómicos nem em registos regulares. Diga, justificando, a que categoria corresponde cada execução.*

Considere o seguinte algoritmo para concretizar um registo regular $(1, N)$.

Implements:
$(1, N)$RegularRegister (on-rreg).

Uses:
BestEffortBroadcast (beb);
PerfectPointToPointLinks (pp2p);
PerfectFailureDetector ($\mathcal{P}$).

upon event $\langle$ *Init* $\rangle$ **do**
value:= 0;
writeSet := $\emptyset$;
correct := Π;

upon event $\langle$ *crash*, p_i $\rangle$ **do**
correct := correct $\setminus$ $\{p_i\}$;

upon event $\langle$ *on-rregRead* $\rangle$ **do**
trigger $\langle$ *on-rregReadReturn*, value $\rangle$;

upon event $\langle$ *on-rregWrite*, val $\rangle$ **do**
trigger $\langle$ *bebBroadcast*, [WRITE, val] $\rangle$;

upon event $\langle$ *bebDeliver*, p_j, [WRITE, val] $\rangle$ **do**
value := val;
trigger $\langle$ *pp2pSend*, p_j, [ACK] $\rangle$;

upon event $\langle$ *pp2pDeliver*, p_j, [ACK] $\rangle$ **do**
writeSet := writeSet $\cup$ $\{p_j\}$;

upon correct $\subseteq$ writeSet **do**
writeSet:= $\emptyset$;
trigger $\langle$ *on-rregWriteReturn* $\rangle$;

Questão 7 *(*1 valor*) Através de um exemplo, demonstre porquê é que este algoritmo não funcionaria caso se usasse um detector de falhas não perfeito.*

Questão 8 *(*2 valores*) Altere algoritmo para funcionar mesmo quando não existe um detector de falhas perfeito.*

Acordo distribuído

Considere o seguinte algoritmo para concretizar o acordo uniforme.

```
00  Implements:
01      UniformConsensus (c).
02
03  Uses:
04      BestEffortBroadcast (beb);
05      PerfectFailureDetector (P).
06
07  upon event ⟨ Init ⟩ do
08      correct := Π; round := 1; decided := ⊥; proposal-set := ∅;
09      for i = 1 to N do delivered[i] := ∅;
10
11  upon event ⟨ crash, p_i ⟩ do
12      correct := correct \ {p_i};
13
14  upon event ⟨ ucPropose, v ⟩ do
15      proposal-set := proposal-set ∪ {v};
16      trigger ⟨ bebBroadcast, [MySet, 1, proposal-set] ⟩;
17
18  upon event ⟨ bebDeliver, p_i, [MySet, r, newSet] ⟩ do
19      proposal-set := proposal-set ∪ newSet;
20      delivered[r] := delivered[r] ∪ {p_i};
21
22  upon (correct ⊆ delivered[round]) ∧ (decided = ⊥) do
23      if round = N then
24          decided := min (proposal-set);
25          trigger ⟨ ucDecide, decided) ⟩;
26      else
27          round := round + 1;
28          trigger ⟨ bebBroadcast, [MySet, round, proposal-set] ⟩;
```

Questão 9 *(*1 valor*) Explique de forma abreviada o funcionamento do algoritmo.*

Questão 10 *(*1 valor*) Considere que substituia a linha 23 por "**if** round = 1 **then**". Através de um exemplo, explique porque é que o algoritmo falharia.*

Questão 11 *(*1 valor*) Considere que substituia a linha 23 por "**if** round = 2 **then**". Através de um exemplo, explique porque é que o algoritmo falharia.*

Questão 12 *(*1 valor*) Tente transmitir a intuição de porque é que o algoritmo funciona ao terminar apenas na ronda N.*

Variantes de consenso

Consider que pretendia resolver o seguinte problema:

Module:

Nome: ConsensusOnMin (com).

Eventos:

Pedido: ⟨ *comPropose*, proposal ⟩: Usado para propor um valor.

Indicação: ⟨ *comDecide*, decision ⟩: Usado para indicar o resultado.

Propriedades:

COM1: *Acordo Uniforme:* Não existem dois processos que decidem valores diferentes.

COM2: *Integridade:* Nenhum processo decide dois valores.

C2: *Validade:* Se um processo decide v, então v foi proposto por algum processo.

COM3: *Minimalidade:* Se um processo decide, temos "proposal $\geq$ decision".

COM4: *Terminação:* Todos os processos correctos decidem alguma vez.

Questão 13 *(*2 valores*) Assuma que possui um detector de falhas perfeito, um serviço de consenso uniforme, e um serviço de difusão fiável. Usando estes componentes, invente um algoritmo para resolver o problema de ConsensusOnMin.*

Replicação usando difusão em grupo fiável

Considere que possui à sua disposição um serviço de comunicação em grupo que oferece os serviços de filiação, sincronia na vista com entrega de mensagens regular ou uniforme e diferentes políticas de ordenação (sem ordem, causal e total).

Usando estes serviços pretende contruir um serviço de venda de bilhetes replicado que oferece duas primitivas: "comprar-bilhete(idcliente, lugar)" e "lista-de-lugares = listar-livres()". Os clientes usam comunicação ponto-a-ponto para contactar uma das réplicas (escolhida de forma aleatória). Caso não obtenham resposta de uma das réplicas, os clientes podem tentar fazer o mesmo pedido a outra réplica.

Questão 14 *(*3 valores*) Apresente o pseudo-código para resolver este problema.*