Sistemas Distribuídos
Aula 12: Difusão Confiável com Detector de Falhas & Uniforme

Prof. Elias P. Duarte Jr.
Universidade Federal do Paraná (UFPR)
Departamento de Informática
www.inf.ufpr.br/elias/sisdis

Sumário

- Revisando a definição e o algoritmo já visto para de *Reliable Broadcast*
- E se tivermos informações sobre falhas? Um algoritmo de *Reliable Broadcast com Detector de Falhas*
- Revisando o conceito da “uniformidade”
- *Uniform Reliable Broadcast:* definição & algoritmo

Difusão Confiável

- Um dos tipos clássicos de difusão (*broadcast*)
- Informalmente: na difusão confiável (*reliable*) todos os processos devem entregar a mensagem, mesmo que o processo origem do broadcast falhe

Difusão Confiável

- Um dos tipos clássicos de difusão (*broadcast*)
- Informalmente: na difusão confiável (*reliable*) todos os processos devem entregar a mensagem, mesmo que o processo origem do broadcast falhe
- Primitiva: *rbcast(msg)*
- Assume falhas crash
- Assume enlaces perfeitos
  - Lembre-se que um enlace perfeito é construído sobre um enlace *fair-loss* e se a origem falha o recebimento da mensagem não é garantido

Propriedades do Reliable Broadcast

- Acordo (*Agreement*): se um processo correto entrega a mensagem *msg*, então todo processo correto *eventually* entrega *msg*
- Validade (*Validity*): se um processo correto envia a mensagem msg usando *rbcast(msg)*, então todos os processos corretos *eventually* entregam *msg*
- Integridade *(Integrity)*
  - Sem-Mensagens-Espúrias *(No creation)*: um processo entrega a mensagem *msg* apenas se ela foi transmitida por um processo do sistema
  - Não-Duplicação *(No Duplication):* nenhuma mensagem é entregue mais de uma vez

Reliable Broadcast

- Como resolver o problema?
- Veja, não tem saída: os processos que entregaram a mensagem vão ter seguir com o broadcast!
- Como fazer isso? Uma alternativa clássica é o seguinte algoritmo

O Algoritmo Básico de Reliable Broadcast

Algoritmo Difusão Confiável

Delivered: conjunto de mensagens entregues

Init: Delivered ← Φ;

Upon rbcast(msg): for all i ∈ S do: send(msg) to i;

Upon receive(msg): if msg **not** in Delivered

then Delivered ← Delivered U {msg};

deliver(msg);

for all i ∈ S do: send(msg) to i;

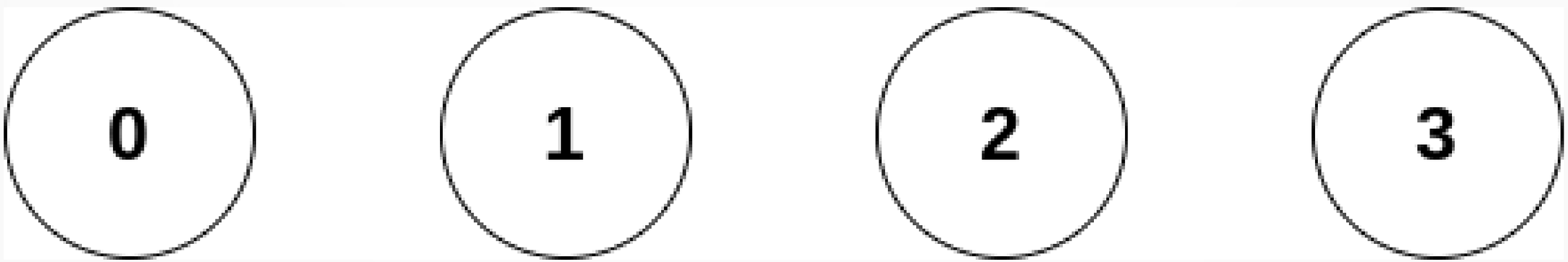
Algoritmo Básico RBCast

- Este algoritmo é **muito** importante
- Funciona mesmo em sistemas assíncronos sujeitos a falhas crash
- Qual o número de mensagens que utiliza?
- Cada processo correto do sistema que recebe a mensagem a transmite para todos os processos do sistema: N^2 mensagens
- Em um sistema com N=100 processos, são 10 mil mensagens...

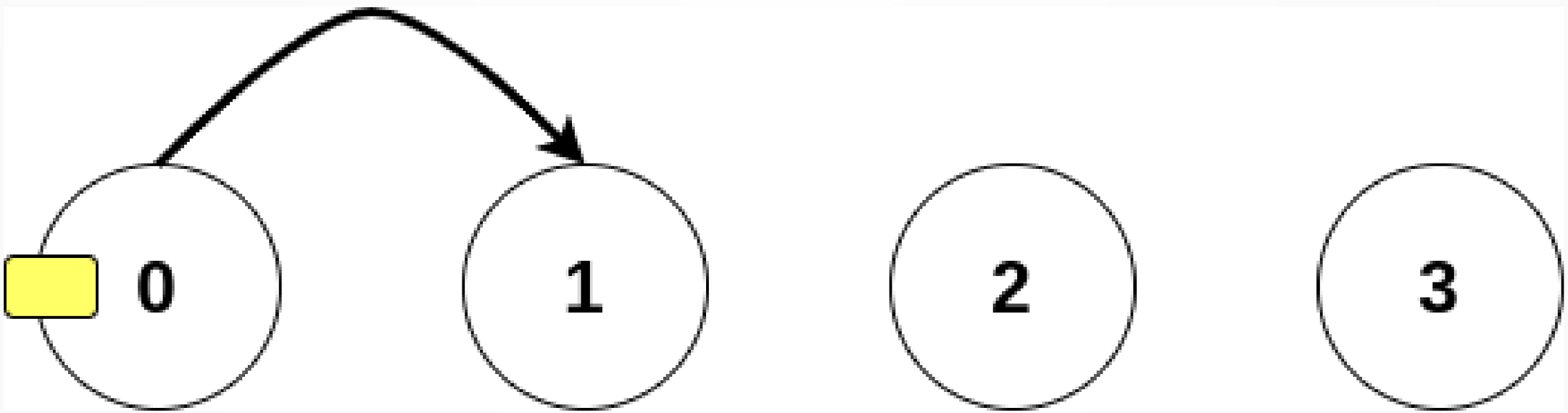
Por que o algoritmo é correto?

- Considere que o processo origem envia a mensagem para 1 único processo e falha
- Sem problemas! Este processo que recebeu vai enviar para todos os outros
- Mas e se envia para apenas 1 processo e falha?
- Sem problemas! Este processo que recebeu vai enviar para todos os outros
- E assim por diante

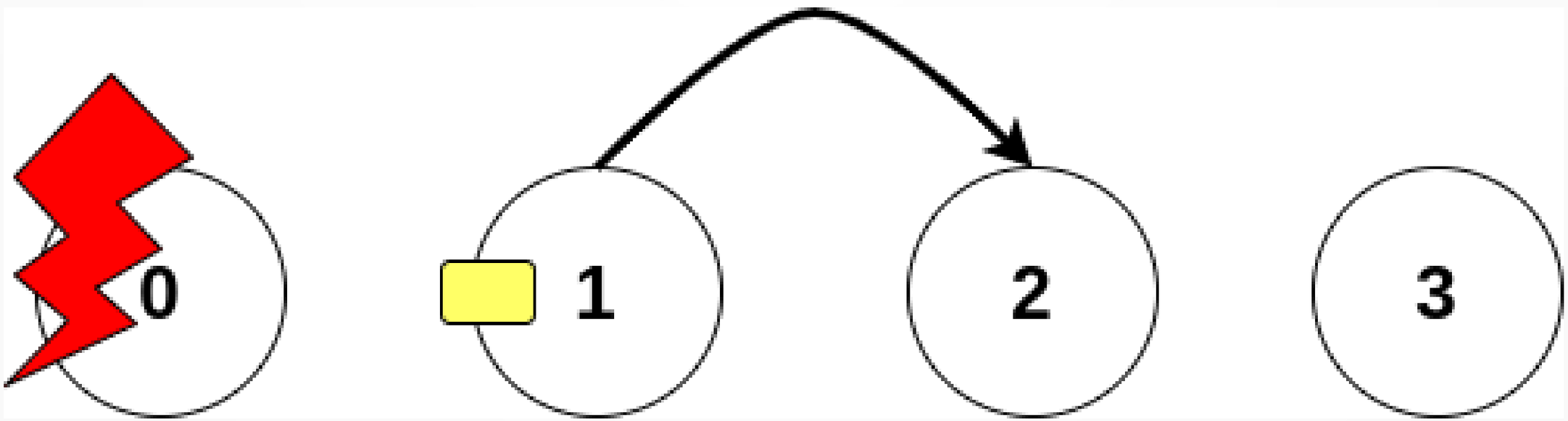
Reliable Broadcast no Extremo



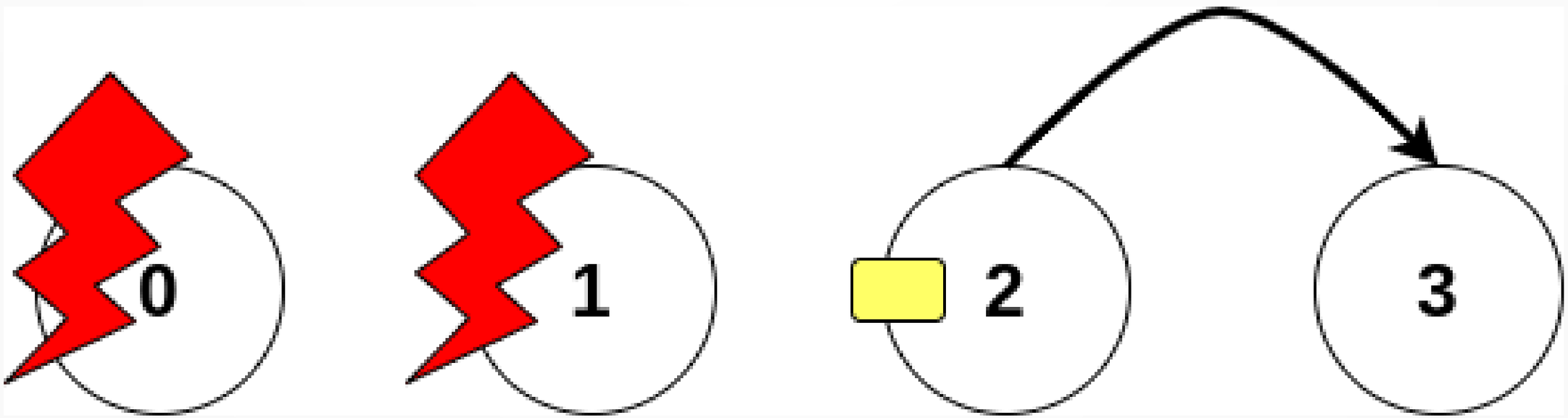
Reliable Broadcast no Extremo



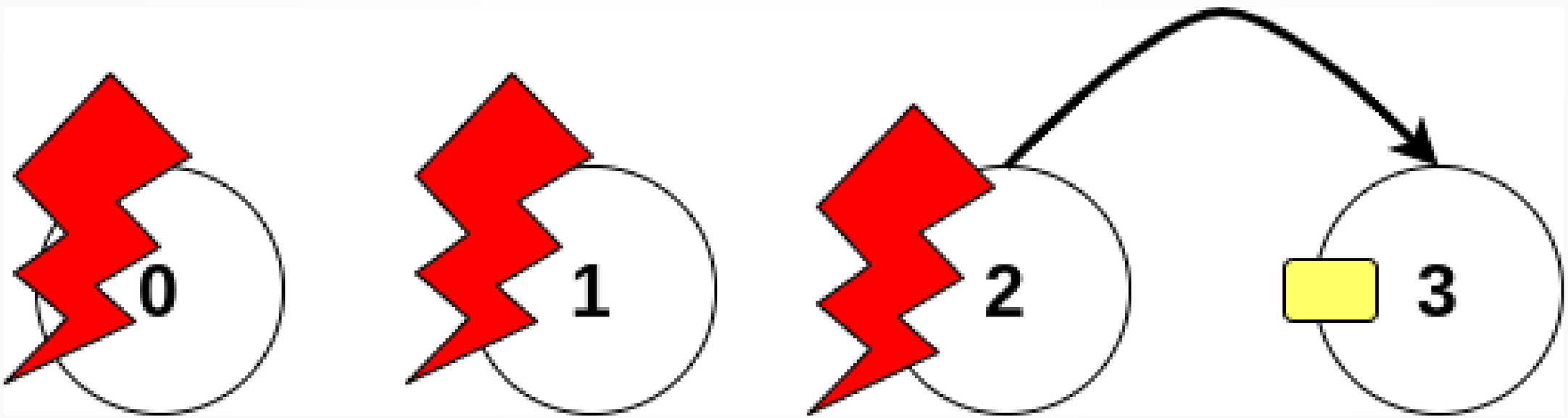
Reliable Broadcast no Extremo



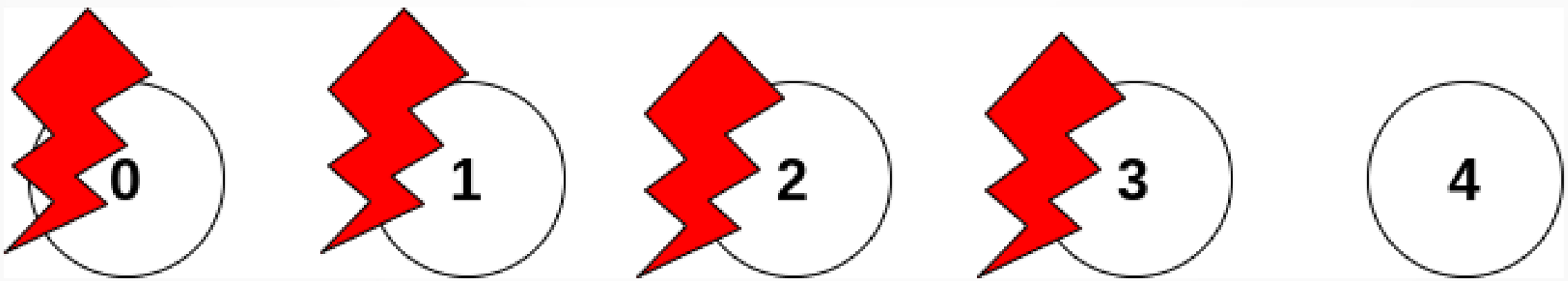
Reliable Broadcast no Extremo



Reliable Broadcast no Extremo



Reliable Broadcast no Extremo



Desempenho

- Como já falamos na outra aula, este algoritmo é extremamente importante!
- Entretanto: requer N^2 mensagens, bastante!
- Como fazer para melhorar este desempenho?
- E se tivermos informações sobre falhas, o que é possível?
- Segue um algoritmo clássico

Reliable Broadcast com Detecção de Falhas – origem do broadcast z

Algoritmo Difusão Confiável com Detector de Falhas

Init: Delivered ← Φ;
Correct ← S;
for all i ∈ S do: MsgFrom[i] ← Φ;

Upon rbcast(msg): for all i ∈ S do: send(msg) to i;

Upon receive(msg): if msg **not** in Delivered
then Delivered ← Delivered U {msg};
deliver(msg);
MsgFrom[z] ← MsgFrom[z] U {msg};
if z not in Correct
then for all i ∈ S do: send(msg) to i;

Upon crashed(z): Correct ← Correct - {z};
for all msg ∈ MsgFrom[z] do: for all i ∈ S do: send(msg) to i;

Difusão Confiável com Detector de Falhas

- Se a mensagem é recebida de um processo correto: não precisa re-encaminhar!
- Mas se o processo origem está falho: reenvia para todos os processos
- Ponto delicado: quando recebe a informação que um processo falhou, tem que transmitir *todas* as mensagens recebidas daquele processo para todos os processos
  - Não se sabe a partir de que ponto o processo falhou!
  - Há estratégias para não mandar todas as mensagens recebidas da origem, mas demandam premissas adicionais

Difusão Confiável com Detector de Falhas

- Se nenhum processo falha: N mensagens! :-D
- Caso contrário: N^2 mensagens

Propriedades do Detector de Falhas

- Quais são as propriedades que o detector de falhas deve satisfazer para ser usado?
- Vamos antes relembrar as duas propriedades básicas:

  → *Completeness* (Completude): o detector de falhas suspeita de processos que efetivamente falharam

  → *Accuracy* (Precisão): o detector de falhas não suspeita de processos que efetivamente estão corretos

Difusão Confiável com Detector de Falhas

- Será que o detector de falhas precisa de completude?

Difusão Confiável com Detector de Falhas: Completude?

- Será que o detector de falhas precisa de completude?
- Sim, pois se um processo que executou rbcast(msg) falhou, ele tem que ser detectado [como falho] para garantir que a msg seja entregue por todos os processos corretos (validade)
- Se um processo não é detectado como tal, é possível que alguns processos corretos entreguem a mensagem, enquanto outros não entreguem

Difusão Confiável com Detector de Falhas: Completude?

- Será que o detector de falhas precisa de completude?
- Sim, pois se um processo que executou rbcast(msg) falhou, ele tem que ser detectado [como falho] para garantir que a msg seja entregue por todos os processos corretos (validade)
- Se um processo não é detectado como tal, é possível que alguns processos corretos entreguem a mensagem, enquanto outros não entreguem

Difusão Confiável com Detector de Falhas: Precisão?

- E será que o detector de falhas precisa satisfazer a precisão?

Difusão Confiável com Detector de Falhas: Precisão?

- E será que o detector de falhas precisa satisfazer a precisão?
- A resposta é: não!
- Veja se não satisfaz a precisão, então pode ser que ocorra uma *falsa* suspeita: um processo correto é suspeito de ter falhado
- Neste caso mais mensagens vão ser transmitidas, mas... tudo bem!
- Em outras palavras: influi no desempenho, não na corretude do algoritmo! :-D

Sobre as Propriedades Necessárias

- Se vocês se lembrarem: a completude é uma propriedade “fácil” de conseguir...
- ... enquanto a precisão é “difícil” (como garantir? não tem jeito!)
- Então, como este algoritmo precisa apenas de completude: excelente!!
- Pode degradar, mas não violar a corretude
- Outro algoritmo muito, muito importante

Uniformidade

- Já mencionamos a definição de algoritmos distribuídos uniformes
- Nestes algoritmos temos que considerar os processo que falham, não apenas os processos corretos!

Uniform Reliable Broadcast

- Na difusão confiável uniforme: se um processo (falho ou correto) entrega uma mensagem, então todos os processos corretos devem entregar a mensagem
- Em particular: o caso que é o foco é aquele em que um processo entrega uma mensagem e falha a seguir
- Pense que “entregar” a mensagem pode significar alterar alguma informação que afeta a consistência do sistema

Propriedades da Difusão Confiável Uniforme

- Acordo Uniforme: se uma mensagem é entregue por um processo x (correto ou falho), então todo processo correto entrega a mensagem
- Integridade e Validade: mantidas
  - Integridade: uma mensagem é entregue no máximo uma vez (não duplicação) e não há mensagens espúrias
  - Validade: se um processo correto z faz broadcast de uma mensagem, então o processo correto x entrega aquela mensagem

Como Implementar?

- Um algoritmo para difusão uniforme não pode deixar um processo entregar uma mensagem e falhar sem todos terem conhecimento disso
- Vamos estudar um algoritmo com detector de falhas

Difusão Uniforme: Algoritmo

Algoritmo Difusão Confiável Uniforme com Detector de Falhas

Init: Delivered ← Φ;
Pending ← Φ;
Correct ← S;
Viram_msg ← Φ;

Upon urbcast(msg): Pending ← Pending U {msg};
for all i ∈ S do: send(msg) to i;

Upon receive(msg) from z: Viram_msg ← Viram_msg U {z}
if msg ∉ Pending
then Pending ← Pending U {msg}
for all i ∈ S do: send(msg) to i;

Upon crashed(i): Correct ← Correct - {i};

Upon (msg ∈ Pending) AND (msg ∉ Delivered) AND (Correct C Viram_msg):
Delivered ← Delivered U {msg};
deliver(msg);

O Algoritmo de Difusão Confiável Uniforme com Detector de Falhas

- A condição chave estabelecida no algoritmo para um processo entregar uma mensagem é saber que todos os demais processos “viram” a mensagem
- Todos os processos retransmitem a mensagem; quando todos os processos corretos tiverem retransmitido: pode entregar!

Propriedades do Detector

- A completude é necessária?

Propriedades do Detector

- A completude é necessária?
- Sim! Pois caso contrário vai ficar esperando para sempre a mensagem de um processo que está falho :-0
- A precisão é necessária?

Propriedades do Detector

- A completude é necessária?
- Sim! Pois caso contrário vai ficar esperando para sempre a mensagem de um processo que está falho :-0
- A precisão é necessária?
- Sim! Pois caso contrário não vai aguardar o ack de um processo correto que pode não receber a mensagem, e assim viola o acordo uniforme
- Completude: progresso & Precisão: *safety*

Um Caminho Sem Detectores

- Uma estratégia para construir a difusão confiável uniforme sem detector de falhas é assumir que sempre há uma maioria (N/2 + 1) de processos corretos
- Neste caso basta receber a mensagem de N/2 processos, formando uma maioria
- Considere o pior caso: destes, N/2 – 1 falham
- Sem problemas! Um deles pelo menos não falha, e retransmite para os demais que também não falham

Conclusão

- Nesta aula primeiro relembramos a definição de reliable broadcast
- Depois relembramos o algoritmo básico: N^2 mensagens
- Vimos um algoritmo para difusão confiável que usa um detector de falhas: N mensagens sem falhas, só precisa de completude
- Estudamos então a definição de difusão uniforme
- Vimos um algoritmo para difusão uniforme que usa um detector de falhas: precisa de completude e precisão
- Uma pincelada em algoritmos baseados em maioria

Referências:

A melhor referência para os diversos tipos de broadcast ainda é o Capítulo 5 do Sape Mullender:

V. Hadzilacos, S. Toueg, “Fault-Tolerant Broadcasts and Related Problems,” **in** Sape Mullender, *Distributed Systems*, Addison-Wesley, 1993.

O BEBcast é definido nosso livro texto.

Obrigado!
Página da Disciplina
Sistemas Distribuídos:
www.inf.ufpr.br/elias/sisdis