Algoritmos e Sistemas Distribuídos

Aula teórica 11

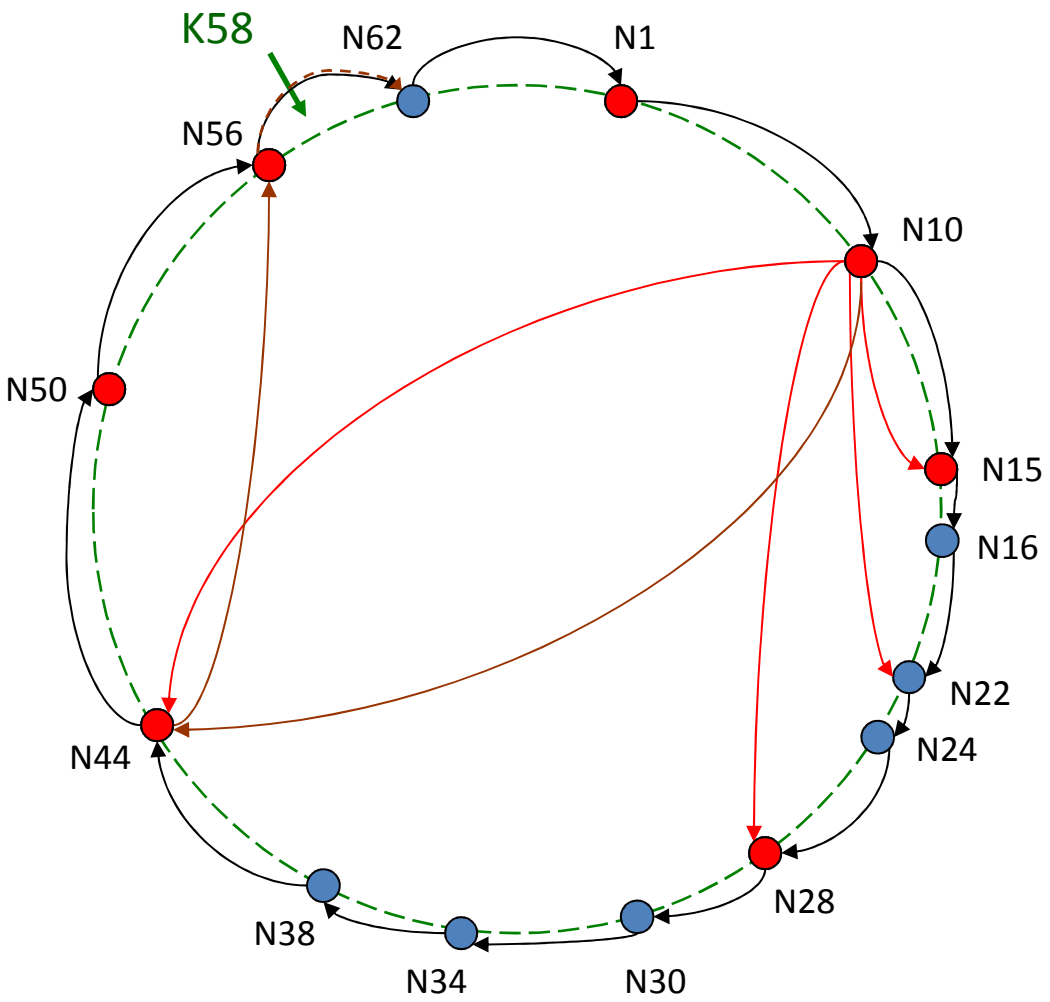
Ano lectivo 2014/2015

Mestrado Integrado em Engenharia Informática

Última aula: p2p routing

- **m = 6**
- **Qual a tabela de fingers do nó com id = 10?**

- **Como se pode encontrar o responsável pela chave 58?**



Última aula: Desafios em p2p

- Controlo da admissão de membros no sistema
- Integridade e disponibilidade dos dados
- Incentivos
- Auto-gestão
- Desconfiança dos ISPs

Hoje: Transações distribuídas

- Objectivo: preservar as propriedades das transações (ACID) num cenário distribuído e com falhas
- Modelo: Base de dados distribuída
  - Transação com vários processos
  - Um coordenador, restantes participantes
- Processos podem ter falhas por crash e recuperar
- Processos têm acesso a armazenamento persistente (para escrever um “log”)

Problema do “atomic commit” (AC): inputs e outputs

- Input de p_i: $vote_i$ in {yes,no}
- Output de p_i: $decision_i$ in {commit, abort}

- $vote_i$ é a visão de cada processo sobre se a transação obedece às propriedades ACID
  - No caso de controlo de concorrência otimista, indica se a transação pode ser serializada na ordem proposta pelo coordenador → leu a versão correta dos dados
  - No caso de locking, indica se todos os locks conseguiram ser adquiridos, ou seja, nenhum fez timeout

Especificação do AC

- AC1 (acordo): Quaisquer dois processos que decidem tomam a mesma decisão
- AC2 (validade, parte 1): Se algum processo começa com o valor “no” então “abort” é a única decisão possível
- AC3 (validade, parte 2): Se todos os processos começam com o valor “yes” e nenhum falhar então “commit” é a única decisão possível
- AC4 (terminação): Se “eventually” todos os processos recuperarem de todas as falhas, então “eventually” todos os processos decidem

Two-phase commit (2PC)

- Resolve o problema do AC
- Pressupõe sistema síncrono
  - Em particular, timeout == crash (embora possa vir a recuperar no futuro)
  - Na prática 2PC pode ser usado num sistema assíncrono mas não garante AC3
- Notar que AC é versão mais fraca do consenso → pode em caso de dúvida fazer "default" para ABORT + terminação menos exigente

2PC

Coordenador C

1. Enviar vote-req a todos participantes

3. Se todos votaram sim então
   $decide_c$ = COMMIT
   enviar COMMIT a todos
   Senão
   $decide_c$ = ABORT
   enviar ABORT aos votantes “yes”

Participante pi

2. Enviar $vote_i$ ao coordenador
   Se ($vote_i$ == no) então
   $decide_i$ = abort;

4. Se recebe COMMIT então
   $decide_i$ = COMMIT
   senão
   $decide_i$ = ABORT

Propriedades já cumpridas?

- Neste momento satisfaz AC1-3
- Problema com AC4?
- No caso de crash, um processo ao recuperar não sabe o que já enviou → em consequência processos podem ficar à espera e não terminar
  - Necessário fazer um log das acções já tomadas
  - Útil fazer um timeout para não esperar por mensagens que nunca chegarão

Acções a tomar em caso de timeout

- Processos esperam nos passos 2,3,4
- Passo 2: pi espera vote-req do coordenador
- Passo 3: coordenador espera votos dos participantes
- Passo 4: pi (que votou sim) espera decisão final
- O que fazer se houver um timeout?

Acções a tomar em caso de timeout

- Processos esperam nos passos 2,3,4
- Passo 2: pi espera vote-req do coordenador
  - pi pode decidir unilateralmente ABORT, porquê?
- Passo 3: coordenador espera votos dos participantes
  - coordenador pode decidir ABORT e enviar ABORT a todos
- Passo 4: pi (que votou sim) espera decisão final
  - pi não pode decidir → corre um protocolo de terminação

Protocolo de terminação

- Uma possibilidade é esperar que o coordenador recupere
  - Funciona pois AC4 só impõe terminação se todos recuperarem
  - Mas pode bloquear a recuperação de forma desnecessária
- Mais rápido: perguntar aos outros participantes
  - Processos que já decidiram enviam decisão, processos que não votaram podem decidir ABORT e enviá-lo
  - E se todos os processos estão à espera?

Necessidade de logging

- Em última análise podem ter de esperar que o coordenador recupere e indique qual foi a sua decisão
  - Coordenador pode estar a recuperar de uma falha, como sabe o que decidiu antes de falhar?
- Esta (e outra) informação é mantida num log

Logging

- Ao enviar VOTE-REQ, coord. escreve START-2PC no log
- Antes de enviar YES, p_i escreve YES no log
- Ao enviar NO, p_i escreve ABORT no log (pode ser após envio)
- Antes de decidir COMMIT e enviar aos participantes, coordenador escreve COMMIT no log
- Ao decidir ABORT, coordenador escreve ABORT no log (pode ser após envio)
- Ao receber a decisão, p_i escreve-a no log

Protocolo de recuperação

- Ao recuperar, se log contem START-2PC:
  - Se log contem COMMIT decidir COMMIT
  - Senão decidir ABORT
- Caso contrário p é participante
  - Se log contem decisão, decidir esse valor
  - Senão, se log não contem voto YES, decidir ABORT
  - Caso contrário, correr protocolo de terminação (perguntar a todos, esperar resposta ABORT de alguém ou resposta definitiva do coordenador)

Segundo teste

- Segunda-feira, dia 1/12, 16:00, sala 127 (ed 2)
- Com consulta, sem dispositivos de comunicação
- Matéria: tudo a partir do primeiro teste **MAIS PAXOS (menos estas considerações gerais sobre a Google, etc.)**
- Boa sorte!