Sistemas Distribuídos como Autómatos

Miguel Pupo Correia

Dep. Informática, Faculdade de Ciências da Universidade de Lisboa
LASIGE, grupo Navigators

CAUL – Maio de 2005

Sumário

1. Sistemas distribuídos
2. Autómatos I/O
3. O problema do consenso
4. Soluções
5. Conclusão

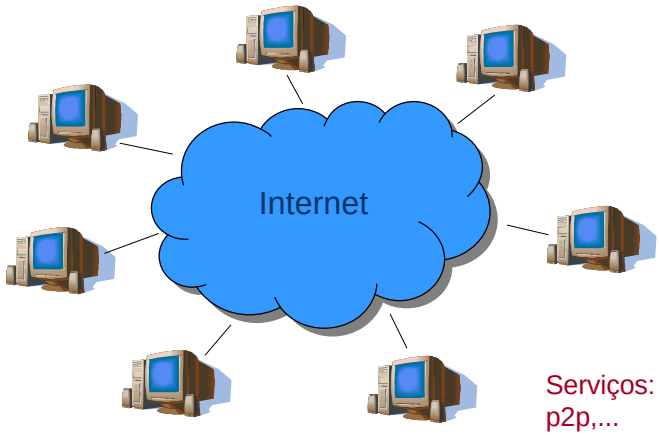
Sumário

1. **Sistemas distribuídos**
2. Autómatos I/O
3. O problema do consenso
4. Soluções
5. Conclusão

Um sistema distribuído é aquele que não o deixa trabalhar por causa da falha de um computador do qual nunca ouviu falar.

- Leslie Lamport

Redes de Computadores



Serviços: web, email, p2p,...

Sistemas Distribuídos

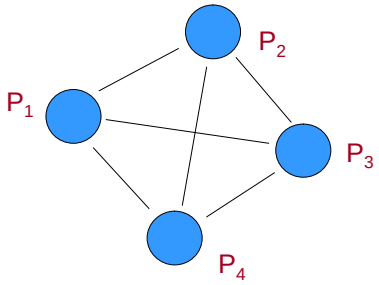
- Sistemas formados por conjuntos de computadores interligados por redes
- Diferença em relação a redes de computad.:
  - ☞Os computadores cooperam para realizar um conjunto de tarefas
  - ☞Há a noção de estado do sistema, partilhado entre todos os computadores
- Exemplos:
  - ☞sistemas de ficheiros distribuídos,
  - ☞sistemas de comércio electrónico,
  - ☞sistemas de trabalho em grupo....

Modelos

- Sistemas distribuídos são complexos a diversos níveis
- Por ex., só o problema da *comunicação em rede* é geralmente dividido em 5 (Internet) ou 7 (ITU) níveis de abstracção!
- Logo, modelos

Modelos topológicos

- Típico e mais simples: processos + canais
  - ☞Comunicação ponto-a-ponto ou por difusão
  - ☞Completamente ligados ou não



Modelos temporais

- Síncrono
  - Há limites conhecidos para os tempos de processamento
  - Há limites conhecidos para os tempos de comunicação
- Assíncrono
  - Não há limites conhecidos para os tempos de processamento
  - Não há limites conhecidos para os tempos de comunicação
- Modelos parcialmente síncronos e extensões

Modelo de Falhas

- Não há falhas
- Paragem de processos
  - Comunicação não falha...
- Processos bizantinos
  - L. Lamport et al., The Byzantine Generals Problem, 1982
- Paragem e omissões na comunicação
- Evitar que o sistema falhe tem enorme importância em muitas áreas (ex. aviação)
  - Confiabilidade / Tolerância a Faltas; Tol. Intrusões

Outros aspectos de modelo

- Processos determinísticos ou não
  - ☞tipicamente sim
- Existência de relógios sincronizados

Algoritmos distribuídos

- Algoritmos executados concorrentemente por diversos processos
- Necessários para concretizar sistemas distribuídos (e não só)
  - ☞Computação paralela, controle de processos tempo-real
- Dependem fortemente do modelo considerado
- Duas dificuldades gerais:
  - ☞Um processo não conhece o estado de outro
  - ☞Têm que lidar com a incerteza do modelo

Sumário

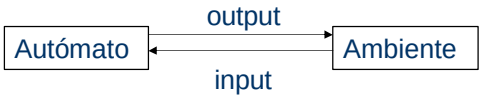
1. Sistemas distribuídos
2. **Autómatos I/O**
3. O problema do consenso
4. Soluções
5. Conclusão

Motivação

- Autómatos I/O: modelo que permite especificar *formalmente* sistemas distribuídos
  - ☞N. Lynch, M. Tuttle 87
- Especificações formais permitem:
  - ☞Fazer provas rigorosas de algoritmos
  - ☞Chegar a resultados sem significado ambíguo (ex: máximos e mínimos)
  - ☞Comparar algoritmos diferentes com precisão
  - ☞Análise de complexidade

Autómatos I/O

- Ideia geral: componentes (lógicas) do sistema são representadas por autómatos I/O
- Autómatos reagem a entradas vindas do *ambiente*
- Três tipos de **acções: *input, output, internas***



Assinatura e acções

- **Assinatura** *S* – partição de um conjunto de acções *acts(S)* em três conjuntos disjuntos:
  - ☞*in(S)*, *out(S)*, *int(S)*
- **Acções externas**:
  - ☞*ext(S) = in(S) U out(S)*

Componentes de um autómato A

- **Assinatura** *sig(A)*
- Conjunto de **estados** *states(A)* – variáveis de estado
- Conjunto de **estados de inicialização** *start(A)* ¹ ø
  - ☞subconjunto de *states(A)*
- **Relação de transição** *trans(A)*
  - ☞*trans(A)*: *states(A)* x *acts(A)* x *states(A)*
- **Partição de tarefas** *tasks(A)*
  - ☞particiona *out(S)* U *int(S)* num conjunto numerável de classes de equivalência
  - ☞cada uma corresponde a um fluxo de execução

Acções activas

- Se (s, p , s') é uma **transição** do autómato A, então a acção p diz-se **activa** em s.
- Duas hipóteses fundamentais do modelo:
  - ☞Todos as acções de input estão sempre activas em todos os estados
  - ☞Nenhuma acção é controlada por mais do que um autómato: nenhum par de processos tem a mesma acção interna ou de output

Execução

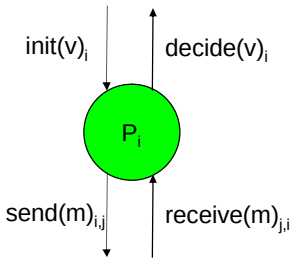
- **Fragmento de execução** de A: sequência finita ou infinita $s_0, p_1, s_1, p_2, \ldots p_n, s_n \ldots$ tal que (s_i, p_{i+1}, s_{i+1}) é uma transição de A, $\forall i$.
- **Execução**: fragmento de execução que começa com um estado de inicialização.
- Denotam-se os conjuntos de execuções de A por *execs(A)* e *finexecs(A)* (finitas).

- **Evento**: ocorrência particular de uma acção.

Exemplos de autómatos

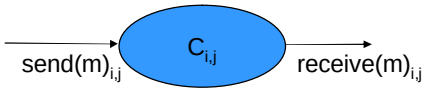
Processo

sig(A)={init(v)$_i$, decide(v)$_i$, send(m)$_{i,j}$, receive(m)$_{j,i}$}



Canal

sig(A)={send(m)$_{i,j}$, receive(m)$_{i,j}$}



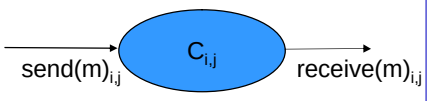
Exemplo – Autómato Canal

- Representação em estilo *pré-condição/efeito*

Assinatura: $send(m)_{i,j}$, $receive(m)_{i,j}$, $m \in M$
Estados: *fila*, uma fila com elementos de *M*, inicialm/ vazia
Transições:

$send(m)_{i,j}$
Ef: pôr *m* no fim da fila

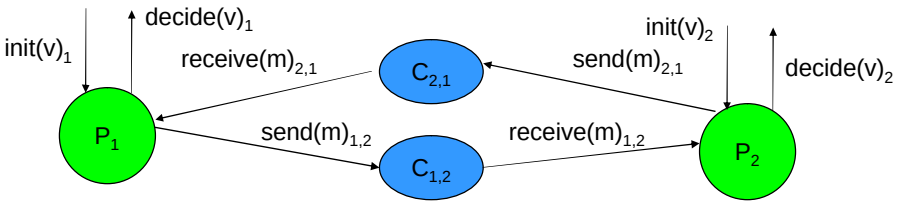
$receive(m)_{i,j}$
Pre: *m* é primeiro da fila
Ef: remover primeiro da fila



Composição

- Operação que permite construir um autómato que modele um sistema complexo usando autómatos que modelem componentes mais simples
- Ligar acções de input e output de mesmo nome
- Quanto um autómato executa a acção de output p, todos os autómatos que têm acções de input p executam p simultaneamente

Exemplo de composição



Compatibilidade

- *Só se podem compôr autómatos compatíveis.*
- Uma colecção numerável de assinaturas $\{S_i\}$ é **compatível** se para todo o i ? j∈I, tivermos
  1. $out(S_i) \cap out(S_j) = \emptyset$,
  2. $int(S_i) \cap acts(S_j) = \emptyset$, e
  3. Nenhuma acção está contida em infinitos conjuntos $acts(S_i)$.
- Uma colecção de autómatos é **compatível** se a colecção das suas assinaturas for compatível.

Composição de assinaturas

- A **assinatura composta** S de uma colecção de assinaturas compatíveis $\{S_i\}_{i \in I}$ é definida como:
  - $in(S) = \bigcup_{i \in I} in(S_i) - \bigcup_{i \in I} out(S_i)$
  - $out(S) = \bigcup_{i \in I} out(S_i)$
  - $int(S) = \bigcup_{i \in I} int(S_i)$
- Acções de input e output:
  - Desaparecem as acções de input às quais são ligadas a acções de output
  - As acções de output mantêm-se todas como tal

25

Composição de autómatos

- A composição $A = \prod_{i \in I} A_i$ de uma colecção numerável de autómatos compatíveis $\{A_i\}_{i \in I}$ é um autómato definido por:
  - $sig(A) = \prod_{i \in I} sig(A_i)$
  - $states(A) = \prod_{i \in I} states(A_i)$
  - $start(A) = \prod_{i \in I} start(A_i)$
  - $trans(A) = \prod_{i \in I} trans(A_i)$
  - $tasks(A) = \bigcup_{i \in I} tasks(A_i)$

26

Execução de uma composição

- Induz a execução dos autómatos componentes
- Dado o autómato $A = \Pi_{i \in I} A_i$
- Dada uma execução $a = s_0\, p_1\, s_1\, p_2 \ldots$ de A
- Seja $a|A_i$ a sequência obtida removendo todos os $p_k\, s_k$ quando p_k não for uma acção de A_i e substituindo os restantes s_k por $s_k[i]$.
- Então $a|A_i \in execs(A_i)$
- *A execução de uma composição é a execução intercalada dos autómatos componentes.*

Especificação de um problema

- Um autómato pode ser visto como uma *caixa negra*, sendo observadas apenas sequências de acções externas: **traços**.
- Um problema pode ser especificado através de uma **propriedade de traço** *P*:
  - ☞*sig(P),* uma assinatura só com acções externas
  - ☞*traces(P),* um conjunto de sequências de acções em *acts(sig(P))*
- O autómato A satisfaz *P* sse:
  - ☞$extsig(A) = sig(P)$ Ù $traces(A) \subseteq traces(P)$

Propriedades

- As propriedades que geralmente se pretende que um sistema verifique são de dois tipos:
- **Propriedade de segurança** (safety):
  - ☞algo que se pretende que seja verificado durante toda a execução; algo “mau” que não pode acontecer.
  - ☞Exemplo: todos os processos *i* que fazem a acção *a(v)* fazem-na com o mesmo valor de *v*
- **Propriedade de vivacidade**:
  - ☞algo “bom” que deve acontecer.
  - ☞Exemplo: o algoritmo termina

Algumas técnicas de prova

- Decomposição modular
  - ☞Pode-se raciocinar sobre uma composição raciocinando sobre autómatos componentes individuais
- Decomposição hierárquica
  - ☞Determinado sistema ou algoritmo é descrito hierarquicamente em diversos níveis de abstracção: do mais elevado e simples ao mais baixo e detalhado; começa-se por provar o de nível mais alto e vai-se provando sucessivamente os abaixo

DIOA

- Um sistema distribuído pode também ser especificado formalmente usando uma **álgebra de processos**
  - ☞CSP, CCS, Cálculo p
- Há uma álgeb. de proc. relacionada com os autómatos I/O: **DIOA**
  - ☞Cada autómato é uma expressão obtida operando autómatos básicos
    - – Segala 92

Sumário

1. Sistemas distribuídos
2. Autómatos I/O
3. **O problema do consenso**
4. Soluções
5. Conclusão

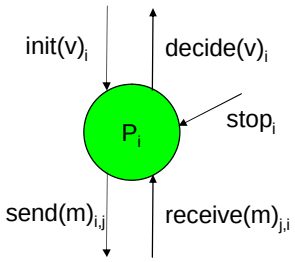
Problema do Consenso

- Informalmente:
  - Há *n* processos, ligados por canais
  - Cada um tem um valor inicial
  - Os processos devem chegar a acordo sobre um desses valores
- Problema importante em sistemas distribuídos pois muitos outros problemas podem ser reduzidos a ele, logo:
  - Algoritmo que resolva consenso pode ser usado para concretizar soluções para esses problemas
  - Resultados teóricos obtidos para consenso aplicam-se a esses problemas

Autómatos

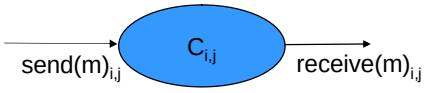
Processo

$sig(A)=\{init(v)_i, decide(v)_i, send(m)_{i,j}, receive(m)_{j,i}\}$



Canal

$sig(A)=\{send(m)_{i,j}, receive(m)_{i,j}\}$



n processos, i ∈ {1,2,...*n*}

v ∈ *V*

Problema do Consenso

segurança

- **Acordo**: Todos os valores decididos são iguais.
  $\forall i,j \ \ decide(v)_i \ \wedge \ decide(v')_j \ \Rightarrow v = v'$ — propr. traço
- **Validade**: Se todas as acções *init* tiverem o mesmo valor *v*, então um processo decide *v*.
  $\forall i \ init(v)_i \Rightarrow \exists j: decide(v)_j$

vivacidade

- **Terminação f-falhas**: Se houver eventos *stop* num máx. de **f** processos, então eventos *decide* ocorrem em todos os processos não falhados.
  $\#\{P_i:stop_i\} \leq f \Rightarrow (\forall i \ decide_i(v) \vee stop_i)$

Esta definição é independente do modelo

Modelo

- Consenso binário: V = {0,1}
- Tempo: assíncrono
- Falhas: paragem

Impossibilidade!

- Terminação 1-falha não pode ser garantida por nenhum algoritmo determinístico com este modelo!
  - ☞ FLP – Fischer, Lynch e Paterson 85
- Provas complicadas mas intuitivamente o problema é que não se consegue distinguir se um processo está lento ou se parou.
- Solução? Mudar qualquer coisa:
  - ☞ Determinístico ? aleatório/probabilístico
  - ☞ Mudar o modelo

Outras impossibilidades

- Resistência (a faltas) máxima: f < n/2
  - ☞ i.e., tolerar 1 falta ⇒ pelo menos 3 processos
- Resistência máxima com faltas bizantinas (modelo assíncrono ou síncrono): f < n/3
- Número mínimo de ciclos de trocas de mensagens: f+1
  (modelo assíncrono ou síncrono)

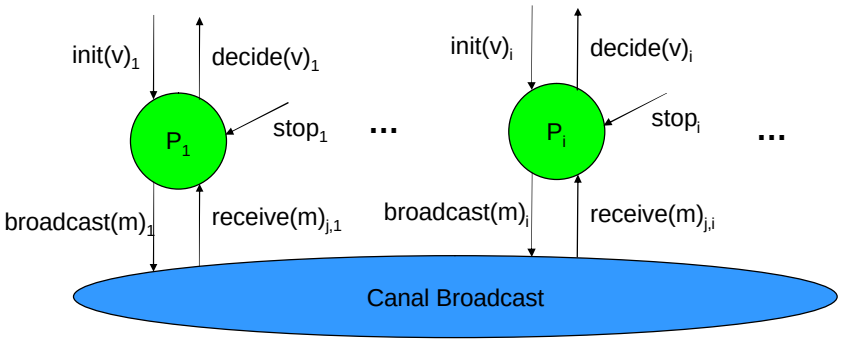
Sumário

1. Sistemas distribuídos
2. Autómatos I/O
3. O problema do consenso
4. **Soluções**
5. Conclusão

Algoritmo Aleatório

- M. Ben-Or 83
- Contorna a impossibilidade FLP por não ser determinístico: a terminação é garantida com probabilidade crescente
- Todo o processo tem um oráculo aleatório
- Consenso binário: V = {0,1}
- Tempo: assíncrono
- Falhas: paragem (max. *f*)
  - ☞tem outro semelhante para bizantinas
- Resistência: $f<n/3$ (sub-óptimo)

Composição de autómatos



Esboço do algoritmo

Processo P_i:

- variáveis locais *x* e *y* inicialmente a *null*
- *init(v)*$_i$ causa *x*? *v*
- executa uma série de fases (1,2,...) cada uma com dois ciclos
- executa o algoritmo para sempre

Fase s ³ 1 :

ciclo 1: broadcast (1,s,v), sendo *v* o valor em *x*;
espera por (n-f) mensagens (1,s,*)
se todas tiverem o mesmo *v*, *y*? *v* ; caso contrário *y*? *null*

ciclo 2: broadcast (2,s,v), sendo *v* o valor em *y*;
espera por (n-f) mensagens (2,s,*)
se todas tiverem o mesmo *v* ¹ *null*, *decide(v)*$_i$ e *x*? *v*
se pelo menos (n-2f) tiverem o mesmo *v* ¹ *null*, *x*? *v*
c.c. *x*? *0 ou 1 com probabilidade 1/2*

Terminação

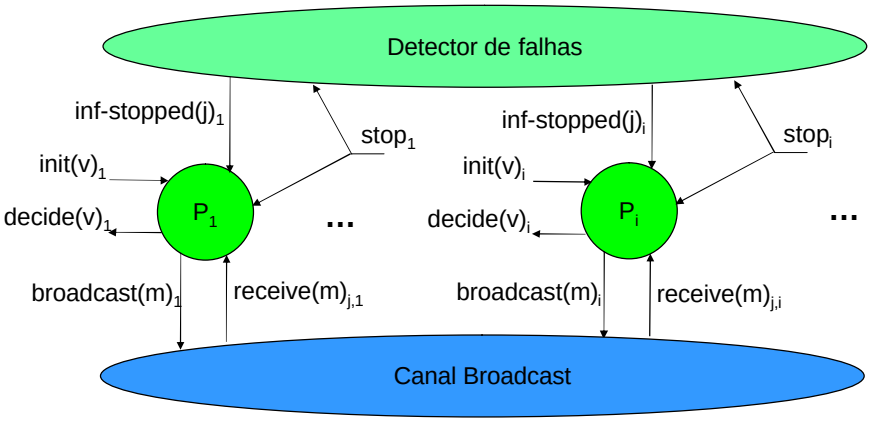
- Esboço de prova:
  - com certa probabilidade todos os processos não parados escolhem o mesmo valor *v* para *x* no ciclo 2 (fase *s*)
  - quando tal acontece todos fazem broadcast de *v* e:
    - todos recebem (n-f) mensagens (1,s,*v*)
    - todos fazem broadcast de (2,s,v) no ciclo 2
    - todos recebem (n-f) mensagens (2,s,*v*)
    - todos fazem *decide(v)*$_i$ e terminam
- Complexidades
  - nº de ciclos esperado: $2^{n-1}+1$
  - nº de broadcasts esperado: $(2^{n-1}+1)n$

Faltaria provar Acordo e Validade

Algoritmo com Detector de falhas

- Modelo extendido com um oráculo *detector de falhas* que dá "dicas" sobre os processos falhados
- Geralmente os detectores considerados não são fiáveis: as dicas podem estar certas ou erradas
- Os detectores incluem algum grau de sincronia, ou seja, não são concretizáveis no modelo assíncrono
- Chandra e Toueg 91
- O algoritmo que vamos ver - Lynch 96 (*f<n/2*)
  - usa um detector de falhas perfeito (não se engana)
  - acções de input: *stop*$_i$
  - acções de output: *inf-stopped(j)*$_i$

Composição de autómatos



Esboço do algoritmo

- Cada processo P_i tenta estabilizar dois dados:
  - vector *val* indexado por {1,...n}, com valores em $V \cup \{null\}$; $val(j) = v \in V$ significa que P_i sabe que o valor inicial de P_j é v
  - conjunto *stopped* com índices dos processos que P_i sabe que pararam (acção $inf\text{-}stopped(j)_i$)
- Inicialmente P_i faz broadcast do seu valor inicial v
- P_i vai actualizando *val* e *stopped*;
  sempre que mudam faz broadcast de *(val,stopped)*
- Quando P_i recebeu mensagens de todos os processos não falhados com *(val,stopped)* iguais ao seu, decide o valor inicial do processo (não falhado) de menor índice

Detector mais fraco

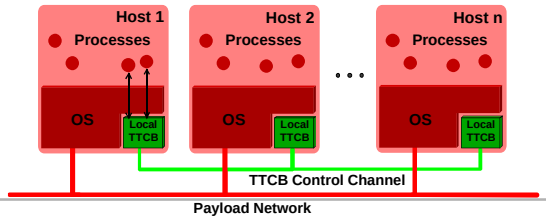
- Qual é o detector de falhas mais fraco que permite resolver consenso?
  - Chandra, Hadzilacos e Toueg 92
- Detector não fiável $\Diamond\mathcal{W}$
  - Completude fraca: a partir de certo instante algum processo correcto suspeita de todos os processos parados.
  - Precisão eventualmente fraca: a partir de certo instante há um processo correcto que não é suspeito de nenhum processo correcto.

Mais soluções

- **Sincronia parcial**
  - Há tempos máximos de processamento e comunicação mas são desconhecidos; ou
  - A partir de certo instante *T* são satisfeitos tempos máximos conhecidos de processamento e comunicação.
    - Dwork et al. 84

Mais soluções

- **Wormholes**
  - Sistema é estendido com um oráculo que oferece um conjunto de serviços. Exemplos:
  - Serviços de tempo: detecção de falhas temporais
  - Serviços "bizantinos": acordo sobre valor "pequeno"
    - Veríssimo et al. 00, Correia et al. 02



Sumário

1. Sistemas distribuídos
2. Autómatos I/O
3. O problema do consenso
4. Soluções
5. **Conclusão**

Conclusão

- Investigação em sistemas distribuídos tem problemas interessantes e que exigem tratamento rigoroso
- Autómatos I/O
- Metodologia semelhante, p.ex., à Física:
  - ☞modelos
  - ☞ferramentas matemáticas para raciocinar nesses modelos: autómatos, álgebras de processos,...
- mas a Informática é uma “ciência do artificial”
  - ☞não só compreender mas também fazer
  - ☞algoritmos têm um papel muito importante

- Obrigado. Perguntas?

- Página pessoal:
  http://www.di.fc.ul.pt/~mpc

- Grupo Navigators:
  http://www.navigators.di.fc.ul.pt/