Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial

Fred B. Schneider

1990

Roteiro

- Introdução
- Máquinas de Estado
- Tolerância a Falhas
- Máquinas de Estado Tolerantes a Falhas
- Reconfiguração do Sistema
- Conclusões

Introdução

- Replicação de serviços para tolerar falhas
- Réplicas precisam ser coordenadas a fim de produzirem o resultado esperado
- Máquinas de estados: *framework* para entender e projetar protocolos de gerenciamento de réplicas
- Resultado: implementação eficiente de réplicas

Máquina de Estado (sm)

- O que é uma sm?
  - Modelo composto de variáveis de estado e comandos
  - Variáveis armazenam seu estado
  - Comandos modificam o seu estado e/ou produzem uma saída
    - Comandos são determinísticos
    - Comandos são atômicos

Exemplo

```
memory: state_machine
        var store : array [0..n] of word

    read: command(loc : 0..n)
          send store[loc] to client
          end read;

   write: command(loc : 0..n , value : word)
          store[loc] := value
          end write
        end memory
```

Máquina de Estado (sm)

- Um cliente de uma sm faz uma **requisição** para executar um comando
  - <sm.command, args>
- Saídas são enviadas para *atuadores*, dispositivos externos ou clientes

Exemplo

```
mutex: state_machine
        var user: client_id init Φ;
            waiting: list of client_id init Φ;

  acquire: command
            if user = Φ → send OK to client;
                          user := client
            □ user ≠ Φ → waiting := waiting ∘ client
            fi
            end acquire

  release: command
            if waiting = Φ → user := Φ
            □ waiting ≠ Φ → send OK to head(waiting);
                            user := head(waiting);
                            waiting := tail(waiting)
            fi
            end release
    end mutex
```

Ordenação das Requisições

- Obedece à relação de causalidade (ordem parcial)
  - O1: Requisições feitas por um mesmo cliente a uma dada sm são processadas na ordem em que foram feitas
  - O2: Numa mesma sm, se uma requisição *r*, de um cliente *c*, pode ter causado a requisição *r'*, de um cliente *c'*, então a sm processa *r* antes de *r'*.
- Devido ao atraso de comunicação, O1 e O2 não implicam que a sm irá processar requisições na mesma ordem de emissão ou de recebimento (influência do IPC)

Semântica de uma sm

- Saídas de uma máquina de estado são **completamente** determinadas pela **seqüência de requisições** que ela processa, independentemente de tempo ou qualquer outra atividade no sistema (IPC)

Semântica de uma sm

- Qualquer aplicação pode ser descrita em termos de máquinas de estado e clientes?
  - Não. É necessário satisfazer a caracterização semântica.
    - Exemplo: estrutura *pc* como um único comando (leitura do sensor *s*, avaliação de *F(s)*, e escrita no atuador). Se leitura do sensor for *time-varying* então *pc* não satisfaz caracterização.
  - Isso não representa uma restrição!

Tolerância a Falhas (TF)

- Componente falho
  - É aquele cujo comportamento não atende sua especificação
- Falhas bizantinas
  - Falhas arbitrárias
- Falhas Fail-stop
  - O componente muda para um estado que permite os demais componentes detectarem que ele falhou e, então, pára.

Tolerância a Falhas (TF)

- Sistema tolerante a *t* falhas
  - Satisfaz sua especificação dado que não mais do que *t* componentes falham durante algum intervalo de interesse
  - Formas para especificar TF:
    - Medida da TF suportada pela arquitetura do sistema

      (MTBF, MTTF, MTTR, R(t), ...)

      Depende das características dos componentes (λ)
    - Número máximo (*t*) de falhas de componentes que podem ser toleradas

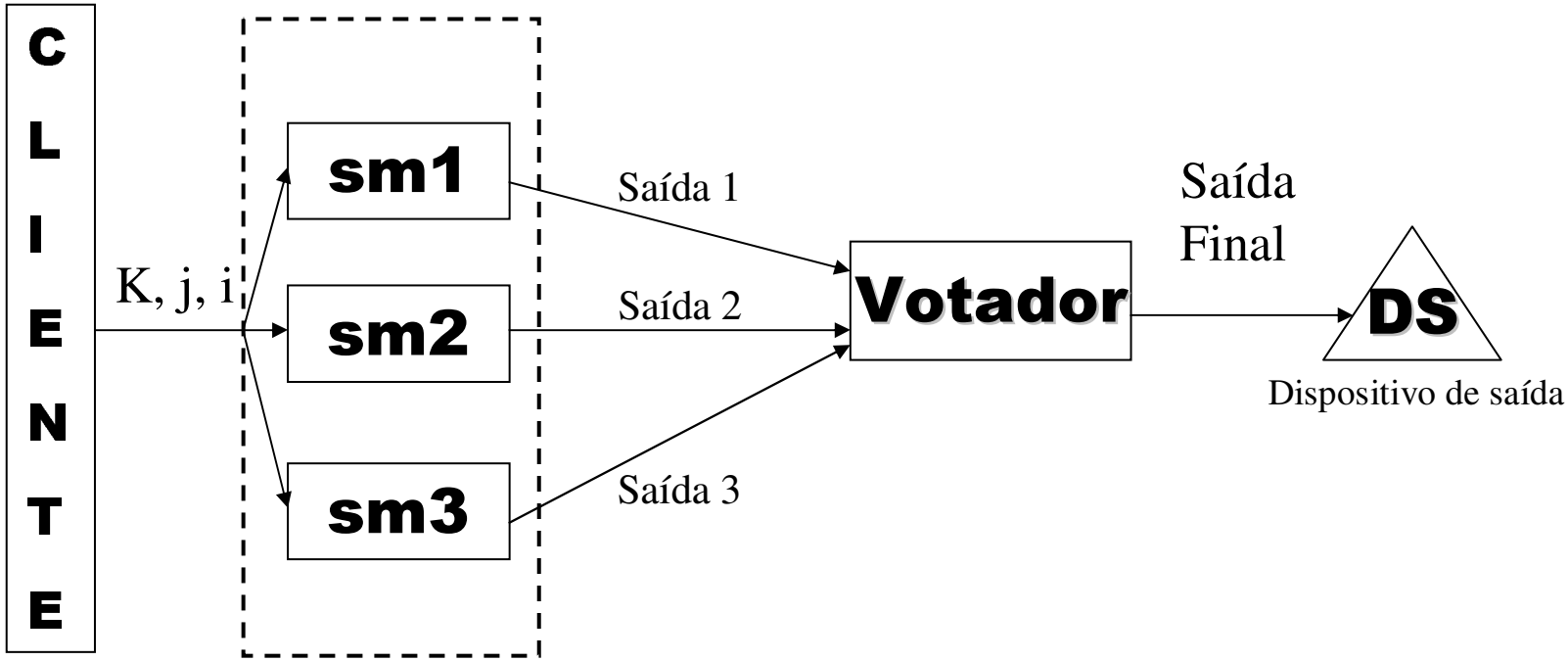
E então?

- Como usar máquinas de estado para atender a um sistema que precisa tolerar *t* falhas?

Máquinas de Estado Tolerantes a Falhas

- Versão TF de uma máquina de estado
  - Replicação da máquina de estado em processadores distintos
  - Cada réplica executa as mesmas requisições na mesma ordem
  - Saídas iguais
- Hipótese: falhas independentes
- Benefício da hipótese: pode-se combinar as saídas das réplicas corretas (votador)

Máquinas de Estado Tolerantes a Falhas



Máquinas de Estado Tolerantes a *t* Falhas

- Na presença de falhas bizantinas, 2t+1 réplicas são necessárias
- Na presença de falhas *fail-stop*, t+1 réplicas são suficientes
- Ponto chave: coordenar as réplicas

Coordenação de Réplicas

- CR: todas as réplicas recebem e processam a mesma seqüência de requisições.
  - Acordo – toda réplica correta recebe toda requisição
  - Ordenação – toda réplica correta processa requisições recebidas na mesma ordem relativa
- É possível burlar a CR desde que considerações adicionais sobre os tipos de requisições e/ou falhas possam ser feitas
  - Requisições *read-only* e processadores *fail-stop* dispensam acordo
  - Requisições comutativas dispensam ordenação.
    - Exemplo: requisições de voto (um por pessoa)

sm para eleição

```
tally: state_machine
       var votes : array[candidate] of integer init 0

  cast_vote: command(choice : candidate)
             votes[choice] := votes[choice]+1;
             if votes[choice]≥MAJ → send choice to SYSTEM;
                                    halt
             [] votes[choice]<MAJ → skip
             fi
             end cast_vote
       end tally
```

Figure 4.1. Election

Acordo

- Protocolos se baseiam num transmissor
  - IC1: todos os processadores corretos concordam sobre o mesmo valor
  - IC2: se o transmissor é correto, então todos os processadores corretos usam seu valor como o valor no qual eles concordam
- Reliable broadcast é um exemplo de protocolo que garante acordo

Ordenação e Estabilidade

- Ordenação
  - Atribuição de uids a requisições
  - Réplicas processam as requisições de acordo com uma ordenação total baseada em uids
  - O1 e O2 tem que ser satisfeitas
    - uid(r) < uid (r´)

Implementação OeE

- Uma réplica processa, seguidamente, a **requisição estável** com o menor uid.
  - Uma requisição é estável em sm_i uma vez que nenhuma requisição de um cliente correto que contenha uid menor é recebida posteriormente por sm_i

Implementação OeE

- Relógios Lógicos
  - sistemas assíncronos + falhas fail-stop
- Relógios Físicos
  - Sistemas síncronos + falhas bizantinas

Relógios Lógicos (RL)

- LC1: T_p é incrementado depois de cada evento no processo p
- LC2: Ao receber uma mensagem com timestamp τ, p atualiza T_p tal que $T_p = \max(T_p, \tau) + 1$

uid(r)=	$T_p(e)=p.T_p$,	e=r

Teste de Estabilidade baseado em RL

- Canais FIFO + Detecção de falhas fail-stop
  - TS: Cada cliente envia requisições periódicas para sm. r é estável na réplica sm_i se uma r´com timestamp maior é recebida por sm_i a partir de cada cliente que executa num processador correto

Teste de Estabilidade baseado em RL

- Prova
  - C é correto
    - Relógios lógicos + canal FIFO → $uid_c(r) < uid_c(r')$
  - C falhou
    - Detecção de falhas fail-stop → nenhuma requisição de c será recebida por sm_i

r é estável

Relógios Físicos

- uid(r)=p.T_p(e), e=r
- Para satisfazer O1 e O2
  - Apenas uma requisição a cada *tick* de relógio
  - O grau de sincronização δ é menor que o atraso mínimo de entrega da mensagem

Relógios Físicos

- Seja Δ uma constante tal que r não pode ser recebida num tempo maior que uid(r)+Δ
- Por definição, quando $T_p=\tau$, p não pode receber r subseqüente tal que uid(r)<τ-Δ

Relógios Físicos

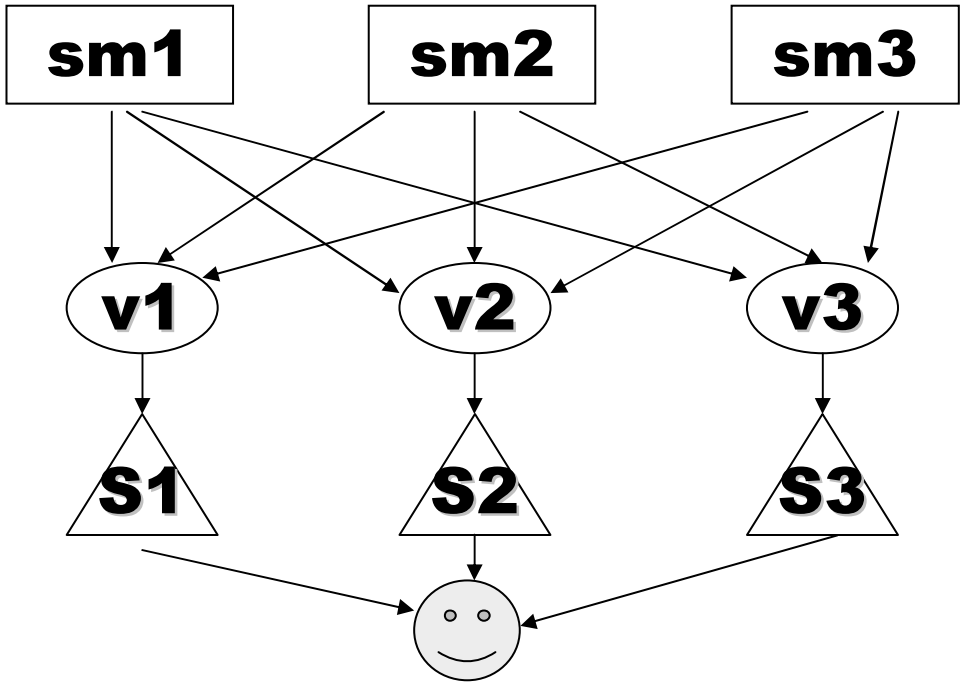
- TS I: r é estável em sm_i, que executa em p, se o relógio local de p é τ e uid(r)<τ-Δ
  - Só é possível concluir que r é estável após Δ
- Otimização
  - TS II: É possível desconsiderar Δ desde que uma requisição com uid maior seja recebida de cada cliente
    - Só funciona quando não ocorrem falhas

E o votador?

- Um único **votador** pode comprometer o objetivo de tolerar t falhas
  - Votador é um ponto único de falha
- Soluções para evitar isso dependem de onde a saída será utilizada

Tolerando Falhas de Dispositivos de Saída

- Saídas usadas fora do sistema
  - Replicar o votador
  - Replicar os dispositivos de saída



Tolerando Falhas de Dispositivos de Saída

- Saídas usadas dentro do sistema (cliente)
  - O próprio cliente pode combinar as saídas produzidas pelas réplicas
    - Votador é parte do cliente
  - Falha do votador → falha cliente
    - Uma saída incorreta do votador é irrelevante

Reconfiguração

- Um grupo de réplicas pode tolerar MAIS do que t falhas se for possível reconfigurá-lo dinamicamente
  - Remoção de réplicas que executam em processadores falhos
  - Adição de réplicas em processadores corrigidos

Reconfiguração

- Para produzir uma saída correta, o grupo de réplicas tem que ter uma configuração tal que $P(\tau) - F(\tau) > Enuf$, $\tau \geq 0$
  - *Enuf* = $P(\tau)/2$ se ocorrem falhas bizantinas
  - *Enuf* = 0, se apenas falhas fail-stop são possíveis

Reconfiguração

- Mais de t falhas bizantinas → remoção de processadores falhos antes que a condição anterior seja violada
- Mais de t falhas fail-stop→ adição de processadores corretos antes que a condição anterior seja violada

Conclusões

- Quando máquinas de estado são utilizadas para prover serviços tolerantes a falhas em sistemas distribuídos, várias questões precisam ser resolvidas
- Relógios lógicos, relógios físicos, consenso, tudo isso é, de fato, indispensável para implementar esses serviços. Funciona!!!